Bill 'MrExcel' Jelen • Tracy Syrstad

VBA e macros
para Microsoft® Office
Excel 2007

Tradução

Sandra Figueiredo e Edson Furmankiewicz

Revisão Técnica

Gustav Schmid e Edson Rosa
Docware Traduções Técnicas

São Paulo
Brasil Argentina Colômbia Costa Rica Chile Espanha
Guatemala México Peru Porto Rico Venezuela

Tradução autorizada a partir da edição original em inglês, *VBA and Macros for Microsoft® Office Excel 2007*, de Bill Jelen, publicada pela Pearson Education, Inc, sob o selo Prentice Hall PTR.

Diretor editorial

Roger Trimer

Gerente editorial

Sabrina Cairo

Editor de livros profissionais

Marco Pace

Revisão de texto

Sandra Scapin

Capa

Rafael Mazzo

Editoração eletrônica

Docware Traduções Técnicas

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Jelen, Bill

VBA e macros para o o Microsoft® Office Excel 2007 / Bill Jelen ; tradução Sandra Figueiredo e Edson Furmankiewicz ; revisão técnica Gustav Schmid e Edson Rosa. -- São Paulo : Pearson Prentice Hall, 2009

Título original: VBA and Macros for Microsoft Office Excel 2007.
ISBN 978-85-7605-195-4

1. Microsoft Office Excel 2007 (Programa de computador) 2. Visual Basic (Linguagem de programação para computador) I. Schmid, Gustav. II. Rosa, Edson. III. Título.

08-09821 CDD-005.369
-055.133

Índices para catálogo sistemático

1. Microsoft Office Excel 2007 : Computadores : Programas : Processamento de dados 005.369
2. Visual Basic : Linguagem de programação : Computadores : Processamento de dados 005.133

2008
Direitos exclusivos para a língua portuguesa cedidos à
Pearson Education do Brasil,
uma empresa do grupo Pearson Education
Av. Ermano Marchetti, 1435
CEP: 05038-001 – São Paulo – SP
Tel.: (11) 2178-8686 – Fax: (11) 3611-0444

Sumário

Autores

Bill Jelen, Excel MVP e MrExcel, trabalha com planilhas desde 1985 e lançou o site Web MrExcel.com em 1998. Bill foi convidado mais de 50 vezes para participar do Call for Help juntamente com Leo Laporte e produziu mais de 250 episódios diários para seu podcast de vídeo, Learn Excel from MrExcel. Ele é o apresentador do DVD Total Training's Excel 2007 Advanced e também gosta de apresentar seus seminários sobre o Excel, que podem durar entre uma e quatro horas, em qualquer lugar onde contadores ou usuários do Excel estejam presentes. Antes de criar o MrExcel.com, Jelen passou 12 anos trabalhando como analista financeiro em departamentos de contabilidade, finanças, marketing e operações para uma empresa pública de US$ 500 milhões. Ele vive perto de Akron, Ohio, com sua esposa, Mary Ellen, e os filhos, Josh e Zeke.

Tracy Syrstad lembra-se do penoso caminho para dominar a curva de aprendizagem do VBA ao desenvolver aplicativos para uso pessoal e para colaboradores no seu antigo trabalho. Agora, como gerente da equipe de consultoria de projetos para o MrExcel, ela gosta de ajudar os clientes a desenvolver soluções personalizadas para suas situações únicas, observando as inúmeras maneiras como as pessoas utilizam o Excel e outros aplicativos do Microsoft Office.

Agradecimentos

Obrigado a Tracy Syrstad por ser uma excelente co-autora e por fazer um ótimo trabalho de gestão de todos os projetos de consultoria no MrExcel. Mala Singh é o assistente mais inteligente de criação de gráficos que conheço e fez um exame minucioso do capítulo sobre como criar gráficos com o VBA. Jerry Kohl trouxe inspiração a muitas das idéias deste livro.

Agradeço a todos que fizeram perguntas na aula sobre Power Macros na University of Akron. Vocês ajudaram a aprimorar esta segunda edição. Obrigado a todos os leitores do MrExcel.com, MVPs e clientes. Agradeço a Chad Rothschiller na Microsoft por me ensinar tudo o que precisa ser conhecido sobre o Excel XML. Agradeço a Dave Gainer, Steve Zaske, Eric Patterson e Joe Chirilov da Microsoft. Agradeço a Pam Gensel pela lição sobre a primeira macro e a Robert F. Jelen por ser meu primeiro fã de programação. Robert K. Jelen, Emil T. Hoffman e Khalil F. Matta foram as influências iniciais que moldaram minha decisão de começar a programar.

Agradeço a Leo Laporte e a todos do The Lab with Leo e a Craig Crossman por serem excelentes jornalistas técnicos.

Agradeço a Dan Bricklin e Bob Frankston por inventarem a planilha de computador. Muito obrigado a Mitch Kapor pelo Lotus 1-2-3. Como todas as outras pessoas que usam computadores como uma atividade profissional, sou enormemente grato a esses três pioneiros.

Agradeço a Barb Jelen por manter o MrExcel em funcionando enquanto eu escrevia. Lora White é nossa gerente administrativa e criou as capturas de tela deste livro em longas viagens de carro. Como sempre, muito obrigado às centenas de pessoas que responderam a 30 mil perguntas sobre o Excel feitas por ano no quadro de mensagens do MrExcel. Agradeço a Duane Aubin, Wei Jiang, Suat Ozgur, Nate Oliver e Jake Hildebrand por suas perícias em programação.

Na Pearson, Loretta Yates é uma impressionante editora de aquisições. Agradeço a Greg Wiegand, Betsy Harris, Laura Norman, Christian A. Kenyeres e Keith Cline. Agradeço a William Waterside Marrom.

Obrigado a Lavan Alexander, Steve Aronson, Oliver Berghaus, Von Brinkley, Wendy Cooke, Chris Crockett, Christy Dare, Joe Degraauw, Todd Feltner, Sylvia Gomez, Louis Grajeda, Jennie Heskin, Michael Himes, Mary Hulshouser, Steve Hunt, Edward Hyden, Richard Irwin, Therese Klassen, Shirley Miller, George Moore, Michael Moreno, Ali Mozaffari, Kiesha Neil, Jay Nelapudi, Ky Nguyen, Victor Omiyale, Andrea Patzkowsky, Earl Pearce, Courtney Robinson, Juan Rubio, Kim Sabatini, Chris Simmons, Michael Snowden, Marlin Snyder, Laree Snyder, Todd Thornbrough, Derek Truman e Yan Yang pelas sugestões no conteúdo do Capítulo 6 durante o primeiro Data Analyst Boot Camp em Frisco, TX.

Por fim, muito obrigado a Josh Jelen, Zeke Jelen e Mary Ellen Jelen.

— Bill

Agradeço a Juan Pablo González Ruiz por oferecer sua experiência, especialmente nas funções descritas no Capítulo 4. Agradeço também a Daniel Klann, Dennis Wallentin, Ivan F. Moala, Juan Pablo González Ruiz, Masaru Kaji, Nathan P. Oliver, Richie Sills, Russell Hauf, Suat Mehmet Ozgur, Tom Urtis, Tommy Miles e Wei Jiang por suas contribuições para o Capítulo 14.

Agradeço aos MVPs do MrExcel.com por me ajudarem a entender as complexidades do mundo do Excel que compartilhamos. Muito obrigado a Jacob Hilderbrand por sua perícia em automatizar o Word.

Agradeço aos desenvolvedores do DOOM 3 — isso manteve John ocupado enquanto eu trabalhava até tarde da noite. Agradeço a Dani Jaacks por me forçar a fazer uma pausa quando eu precisava.

E, por último, mas não menos importante, agradeço a Bill Jelen. Seu site, MrExcel.com, é um local que milhares acessam para obter ajuda. É também um local em que eu, e outros como eu, têm a oportunidade de ajudar outras pessoas.

Dedicatórias

Dedicado a Josh Jelen.

— Bill

Dedicado a meus pais, Russ e Marcelle.

— Tracy

Introdução

Obtendo resultados com o VBA

Enquanto os departamentos de TI se deparam com um grande volume de solicitações não atendidas, os usuários do Excel descobriram que é possível produzir os relatórios necessários para que eles próprios possam administrar seus negócios utilizando a linguagem de macros *Visual Basic for Applications* (VBA). O VBA permite que você alcance uma enorme eficiência no uso rotineiro do Excel. Isso é bom e ruim. O lado bom é que, sem esperar os recursos de TI, você provavelmente consegue descobrir como importar dados e criar relatórios no Excel. O lado ruim é que agora você fica preso ao processo de criação e importação de relatórios no Excel.

NESTA INTRODUÇÃO

O que há neste livro

Você deu o passo certo ao comprar este livro. Ele pode ajudá-lo a aprimorar a curva de aprendizagem para que você possa escrever suas próprias macros VBA e colocar um fim à trabalheira de gerar relatórios manualmente.

Aprimorando a curva de aprendizagem

Esta introdução fornece uma breve história das planilhas. O Capítulo 1 apresenta as ferramentas e confirma o que você provavelmente já sabe: o programa de gravação de macros não funciona. O Capítulo 2 ajuda a entender a sintaxe absurda do VBA. O Capítulo 3 examina o código e discute como trabalhar eficientemente com intervalos e células.

Quando começar a ler o Capítulo 4, você terá conhecimento suficiente para colocar imediatamente em prática os 25 exemplos das funções definidas pelo usuário neste capítulo.

O Capítulo 5 abrange o poder de looping feito com o VBA. No estudo de caso da Valerie, depois que o programa para criar o primeiro relatório para um departamento foi escrito, levou apenas mais um minuto para empacotar a rotina de relatórios em um loop que criou todos os 46 relatórios.

O Capítulo 6 abrange fórmulas no estilo R1C1. O Capítulo 7 examina o que mudou no Excel VBA em relação ao Excel 2003 e Excel 2007. No passado, era relativamente fácil criar código VBA que poderia ser executado em quaisquer versões recentes do Excel. Infelizmente, com as amplas mudanças no Excel 2007, isso tornou-se bem mais difícil. O Capítulo 8 abrange nomes. O Capítulo 9 apresenta alguns truques excelentes que utilizam a programação de evento. O Capítulo 10 introduz caixas de diálogo personalizadas que você pode utilizar para coletar informações sobre o usuário que utiliza o Excel.

Poder do Excel VBA

Os capítulos 11 a 13 fornecem um exame minucioso sobre como utilizar gráficos, filtros avançados e tabelas dinâmicas. Qualquer ferramenta de automação de relatório dependerá significativamente desses conceitos.

O Capítulo 14 inclui outros 25 exemplos de código projetados para mostrar o poder do Excel VBA. Os capítulos 15 a 18 tratam de visualizações de dados, de consultas à Web, de XML e de como automatizar outro programa do Office, como o Word.

O *material técnico* necessário para produzir aplicativos para outras pessoas

O Capítulo 19 mostra como utilizar arrays para criar aplicativos rápidos. Os capítulos 20 e 21 discutem a leitura e gravação em arquivos de texto e em bancos de dados Access. As técnicas para utilizar bancos de dados Access permitem construir um aplicativo com os recursos de multiusuários do Access e ainda manter a interface amigável do Excel.

O Capítulo 22 analisa o VBA do ponto de vista de um programador do Visual Basic. Ele analisa classes e coleções. O Capítulo 23 discute os tópicos sobre userforms avançados. O Capítulo 24 ensina algumas maneiras criativas de realizar tarefas que utilizam a interface de programas aplicativos do Windows. Os capítulos 25 a 27 lidam com tratamento de erros, menus personalizados e suplementos.

Este livro ensina o Excel?

A Microsoft acredita que o usuário médio do Office usa apenas 10% de seus recursos. Percebi que qualquer leitor deste livro está acima da média. Acredito que o público do MrExcel.com é bem inteligente. Uma pesquisa com 8 mil leitores do MrExcel.com mostra que apenas 42% dos usuários com um nível de conhecimento acima da média utilizam um dos 10 recursos avançados do Excel. Apresento regularmente o seminário Power Excel para contadores. Eles são os usuários que utilizam o Excel rotineiramente 30 a 40 horas por semana. Como de costume, surgem duas coisas em todos os seminários. Primeiro, metade do público se impressiona quando vê a rapidez com que é possível executar tarefas com um determinado recurso (como subtotais automáticos ou tabelas dinâmicas). Segundo, alguém na platéia sempre fornece uma resposta melhor que a minha. Alguém faz uma pergunta, eu respondo e outra pessoa na segunda fileira levanta a mão e dá uma resposta melhor. A razão? Tanto você como eu conhecemos o Excel em detalhes. Entretanto, imagino que, em um capítulo qualquer, talvez 58% das pessoas ainda não tenha utilizado tabelas dinâmicas e talvez um número ainda menor ainda não tenha utilizado o recurso "10 melhores filtros" das tabelas dinâmicas. Antes de mostrar como automatizar algo no VBA, discutiremos brevemente como fazer a mesma tarefa na interface do Excel. Este livro não ensina como criar tabelas dinâmicas, mas fornece dicas de que é possível explorar algo e aprendê-lo em outra parte.

Estudo de caso

Relatórios mensais contábeis

A história a seguir é real. Valerie é uma analista de negócios no departamento de contabilidade de uma corporação de médio porte. Sua empresa recém-instalou um sistema ERP (*Enterprise Resource Planning*) de US$ 16 milhões, o que estourou o orçamento. À medida que o projeto se aproximava do final, o departamento de TI não tinha mais recursos financeiros para produzir o relatório mensal que essa corporação utilizava para fazer um resumo das atividades de cada departamento.

Valerie, porém, estava quase no final do processo de implementação e tentava encontrar uma maneira de ela mesma criar o relatório. Ela entendia que seria possível exportar os dados contábeis a partir do sistema ERP para um arquivo de texto com valores separados por vírgulas. Utilizando o Excel, Valerie conseguiu importar os dados contábeis do sistema ERP para o Excel.

Não foi fácil criar o relatório. Como ocorre em muitas empresas, sempre há exceções nos dados. Valerie sabia que certas contas em um determinado centro de custos precisavam ser reclassificadas como uma despesa. Ela sabia que outras contas precisavam ser excluídas do relatório. Trabalhando cuidadosamente no Excel, Valerie fez estes ajustes. Ela criou uma tabela dinâmica para produzir a primeira seção de resumo do relatório. Ela recortou os resultados da tabela dinâmica e colou-os em uma planilha vazia. Em seguida, criou um novo relatório de tabela dinâmica para a segunda seção do resumo. Depois de quase três horas, ela havia importado os dados e criado cinco tabelas dinâmicas; então, organizou-os em um resumo e formatou primorosamente o relatório com cores diferentes.

Ela tornou-se uma heroína

Valerie entregou o relatório ao seu gerente. O departamento de TI havia comentado com esse gerente que seriam necessários *vários meses* para encontrar uma maneira de criar "esse intricado relatório". Valerie entrou, apresentou o relatório no Excel e tornou-se instantaneamente a heroína do dia. Em três horas, conseguiu fazer *o impossível*. Ela ficou nas nuvens depois de um bem-merecido reconhecimento.

Mais aclamações

No dia seguinte, esse gerente participou da reunião mensal do departamento. Quando os gerentes do departamento começaram a reclamar que não conseguiam receber o relatório a partir do sistema ERP, o gerente pegou o relatório do seu departamento e o colocou na mesa. Os outros gerentes ficaram surpresos. Como ele foi capaz de criar esse relatório? Todos se sentiram aliviados ao saber que alguém tinha decifrado o código. O presidente da empresa perguntou ao gerente de Valerie se o relatório poderia ser criado para cada departamento.

Os elogios transformam-se em apreensão

Certamente podemos ver o resultado disso. Essa empresa tinha 46 departamentos. Isso significava que 46 resumos de uma página tinham de ser produzidos uma vez por mês. Cada relatório exigia importar dados a partir do sistema ERP, rever certas contas, produzir cinco tabelas dinâmicas e depois formatá-las em cores. Valerie levou três horas para produzir o primeiro relatório. Ela descobriu que, depois de pegar o jeito, poderia criar 46 relatórios em 40 horas. Isso era terrível. Valerie tinha uma tarefa a fazer antes de assumir a responsabilidade pelas 40 horas mensais para produzir esses relatórios no Excel.

VBA em nosso socorro

Valerie descobriu minha empresa, a MrExcel Consulting, e explicou a situação. No período de uma semana, consegui produzir uma série de macros no Visual Basic que fazia todas as tarefas triviais. Essa série de macros importava os dados. Revia algumas contas. Criava cinco tabelas dinâmicas e aplicava a formatação em cores. Do início ao fim, todo o processo manual de 40 horas foi reduzido a dois cliques de botão e a aproximadamente quatro minutos.

Agora, você ou alguém na sua empresa provavelmente esteja limitado a fazer tarefas manuais no Excel que poderiam ser automatizadas com o VBA. Estou certo de que *posso entrar em uma empresa* qualquer com 20 ou mais usuários do Excel e encontrar uma situação tão surpreendente quanto à de Valerie.

O futuro do VBA e as versões Windows do Excel

Há quatro anos, circulavam vários boatos de que a Microsoft poderia parar de dar suporte ao VBA. Atualmente, há várias evidências de que o VBA permanecerá nas versões Windows do Excel até 2015. (Esse futuro não é tão certo para a versão Macintosh do Excel.) O Microsoft Office Excel 2007 foi lançado em 30 de janeiro de 2007. A Microsoft informa que, na próxima versão do Excel (o Excel 14), ela vai parar de dar suporte a macros XLM. Essas macros foram substituídas pelo VBA há 14 anos, mas ainda são suportadas. Na cúpula 2005 MVP, os membros da equipe de desenvolvimento do Office previram que o suporte ao VBA continuará por outros 10 a 15 anos. Fala-se até mesmo de uma melhoria para o Visual Basic Editor no Excel 14.

Mesmo assim, podemos ver uma falta de comprometimento por parte da Microsoft com o VBA. O Office 2003 oferecia alguns recursos, como o Research Pane e SmartTags, que só poderiam ser automatizados com o Visual Basic .Net. No Excel 2007, o programa de gravação de macros funciona em aproximadamente 50% dos comandos de criação de gráficos, mas não consegue gravar uma quantidade significativa deles.

As ferramentas que você conhece hoje são boas para os próximos 10 anos. Se a Microsoft decidir descontinuar o VBA em favor do Visual Studio Tools for Office (VSTO) ou de alguma outra ferramenta, provavelmente você será capaz de transferir suas habilidades de codificação a essa nova plataforma.

Versões

Esta segunda edição do *VBA e Macros para Microsoft Office Excel 2007* foi projetada para funcionar com o Excel 2007. Nossa edição anterior abrangia o código para as versões do Excel 97 ao Excel 2003. Em 80% dos capítulos, o código para o Excel 2007 será idêntico àquele das versões anteriores. Há exceções. A Microsoft oferece uma nova lógica de classificação. Os gráficos mudaram completamente. As ferramentas de formatação condicional e visualização de dados no Capítulo 15 são novas. As tabelas dinâmicas mudaram um pouco. Os exemplos de XML no Capítulo 17 só funcionarão no Excel 2003 ou em uma versão mais recente. Embora o Excel para o Windows e o Excel para o Mac sejam semelhantes em relação à interface com o usuário, há várias diferenças quando o comparamos ao ambiente do VBA. Certamente, nada no Capítulo 24 que utiliza a API do Windows funcionará no Mac. Os conceitos gerais discutidos no livro se aplicam ao Mac, mas haverá diferenças. Você pode encontrar uma lista geral dessas diferenças e da maneira como elas se aplicam ao Mac em www.mrexcel.com/macvba.html.

Elementos especiais e convenções tipográficas

As seguintes convenções tipográficas são utilizadas neste livro:

- *Itálico* — Indica novos termos quando eles são definidos, ênfase especial, palavras ou frases em outros idiomas que não o inglês e letras ou palavras utilizadas como palavras
- `Monospace` — Indica partes do código VBA, como nomes de objeto ou método e nomes de arquivo (*filenames*)
- *`Itálico monoespaçado`* — Indica texto de marcador de lugar na sintaxe do código
- **`Bold monospace`** — Indica entrada de usuário

Além dessas convenções de texto, há também vários elementos especiais. Cada capítulo tem pelo menos um estudo de caso que mostra soluções reais a problemas comuns e aplicações práticas dos tópicos discutidos no livro. Além desses estudos de caso, você também verá as seções Novos ícones, Notas, Dicas e Atenção.

NOVO Novos recursos ou recursos que são muito diferentes no Excel 2007 são marcados com esse ícone.

NOTA

As Notas fornecem informações adicionais que estão além do escopo do capítulo e que talvez você ache útil conhecer.

DICA

As Dicas fornecem maneiras rápidas de contornar problemas e técnicas para economizar tempo a fim de ajudá-lo a tornar seu trabalho mais eficiente.

ATENÇÃO

As caixas de Atenção alertam sobre armadilhas que você pode encontrar. Preste atenção a elas, porque esses avisos alertam sobre problemas que podem lhe causar horas de frustração.

Arquivos de código

Como uma forma de agradecer pela compra deste livro, os autores agruparam um conjunto de 50 pastas de trabalho do Excel que demonstram os conceitos aqui apresentados. Essa série de arquivos inclui todos os códigos que se encontram no livro, dados de exemplo, notas adicionais dos autores, além de mais 25 macros de bônus. Para fazer o download dos arquivos de código, visite essa página do livro em www.prenhall.com/jelen_br ou www.quepublishing.com ou www.mrexcel.com/getcode2007.html.

Próximos passos

O Capítulo 1 apresenta as ferramentas de edição do ambiente Visual Basic e mostra por que utilizar o programa de gravação de macros não é uma maneira eficiente de escrever o código de macro VBA.

Libere o poder do Excel com o VBA

1

NESTE CAPÍTULO

O poder do Excel

O *Visual Basic for Applications* (VBA) combinado com o Microsoft Excel é provavelmente a ferramenta mais poderosa disponível para você. Essa ferramenta está nos desktops de 500 milhões de usuários do Microsoft Office e a maioria deles nunca descobriu como aproveitar o poder do VBA no Excel. Utilizando o VBA, você pode agilizar a produção de qualquer tarefa no Excel. Se utilizar regularmente o Excel para produzir uma série de gráficos mensais, o VBA poderá realizar a mesma tarefa para você em questão de segundos.

Barreiras iniciais

Há duas barreiras a um aprendizado bem-sucedido na programação VBA. A primeira é que o programa de gravação de macros do Excel tem falhas e não cria um código viável que possa ser utilizado como modelo. A segunda barreira é que, para muitos que aprenderam uma linguagem de programação como o BASIC, a sintaxe do VBA é terrivelmente frustrante.

O programa de gravação de macros não funciona!

A Microsoft começou a dominar o mercado de planilhas em meados dos anos 1990. Embora ela tenha sido muito bem-sucedida em criar um poderoso programa de planilha para o qual qualquer usuário do Lotus 1-2-3 poderia facilmente migrar, a linguagem de macros era muito diferente. Qualquer pessoa que conseguisse gravar macros no Lotus 1-2-3 e que tentasse gravar algumas no Excel muito provavelmente falharia. Embora a linguagem de programação do VBA da Microsoft seja muito mais poderosa que a linguagem de macros do Lotus 1-2-3, sua principal falha é que o programa de gravação de macros não funciona.

Com o Lotus 1-2-3, você poderia gravar uma macro hoje, reutilizá-la amanhã e ela funcionaria de uma maneira confiável. Quando você tenta a mesma façanha no Microsoft Excel, a macro pode funcionar hoje, mas não amanhã. Fiquei muito frustrado em 1995 quando tentei gravar minha primeira macro no Excel.

O Visual Basic não é como o BASIC

O código gerado pela macro era diferente de tudo o que eu já tinha visto. Dizia-se que isso era coisa do 'Visual Basic'. Tive o prazer de aprender uma dezena de linguagens de programação em diferentes épocas; essa linguagem de aparência esquisita não tinha nada de intuitiva e de maneira alguma parecia com a linguagem BASIC que aprendi na escola.

Para piorar as coisas, em 1995, eu era o responsável pela criação de planilhas no escritório em que trabalhava. Minha empresa tinha acabado de fazer todo mundo migrar do Lotus 1-2-3 para o Excel. Na época, tinha pela frente um programa de gravação de macros que não funcionava e uma linguagem que eu não conseguia entender. Essa não era uma boa combinação de eventos.

Minha hipótese ao escrever este livro é a de que você tem muito talento na produção de planilhas. Você provavelmente sabe mais do que 90 por cento das pessoas do seu escritório. Vou supor que você não é um programador, mas tem algumas noções do BASIC. Isso não é um requisito — na verdade, é uma barreira inicial para que consiga ser um programador bem-sucedido do VBA. É muito provável que você já tenha gravado uma macro no Excel e não tenha ficado satisfeito com os resultados.

A boa notícia — é fácil passar para outro nível na curva de aprendizagem

Mesmo que você já tenha ficado frustrado com o programa de gravação de macros, na verdade isso é apenas um pequeno obstáculo ao longo da aprendizagem para escrever programas poderosos no Excel. Este livro ensinará por que o programa de gravação de macros falha, mas também mostrará como é fácil transformar o código gravado em algo útil. Para todos os ex-programadores em BASIC na platéia, decodificarei essa linguagem de aparência esquisita para que você possa verificar facilmente o código da macro gravada e entender o que está acontecendo.

A boa notícia — o Excel associado ao VBA recompensa os esforços

Embora talvez você tenha ficado frustrado com a Microsoft em relação à sua incapacidade de gravar macros no Excel, a boa notícia é que o Excel VBA é poderoso. Absolutamente tudo que você pode fazer na interface do Excel pode ser reproduzido no Excel VBA a uma velocidade impressionante. Se você criar rotineiramente os mesmos relatórios manualmente, dia após dia e semana após semana, o Excel VBA simplificará significativamente essas tarefas.

Os autores trabalham para a MrExcel Consulting. Nesse papel, ajudamos a automatizar relatórios para centenas de clientes. Em geral, as histórias são semelhantes: o departamento de informações gerenciais tem um volume acumulado de solicitações pendentes há vários meses. Alguém da contabilidade ou da engenharia descobre que é possível importar alguns dados para o Excel e obter os relatórios necessários para administrar o negócio. Isso é um evento libertador — você não mais precisa esperar meses para que o departamento de TI escreva um programa. Entretanto, o problema é que depois de importar os dados para o Excel e receber elogios do seu gerente por ter criado o relatório, você irá se deparar com uma situação em que será necessário produzir o mesmo relatório todos os meses ou todas as semanas. Isso se torna bem entediante.

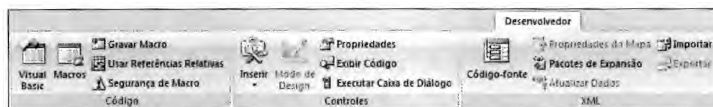
Mais uma vez, a boa notícia é que, com algumas horas de programação VBA, você pode automatizar o processo de criação de relatórios e transformá-lo com alguns cliques de botão. A recompensa é fantástica. Vamos juntos examinar alguns princípios básicos.

Este capítulo mostrará por que o programa de gravação de macros não funciona. Analisaremos um exemplo simples de um código gravado e demonstraremos por que ele funcionará hoje, mas não amanhã. Você verá o código aqui e talvez ainda não o conheça. Sem problemas. A razão deste capítulo é demonstrar o problema fundamental com o programa de gravação de macros. Também abordaremos os princípios básicos do ambiente do Visual Basic.

Conhecendo suas ferramentas — a faixa Desenvolvedor

NOVO Começaremos com uma visão geral básica das ferramentas necessárias para utilizar o VBA. Por padrão, a Microsoft oculta as ferramentas do VBA. Precisamos mudar uma configuração nas opções do Excel para acessar a faixa Desenvolvedor.

No ícone Office, selecione Opções do Excel. A terceira configuração na categoria Mais Usados é Mostrar Guia Desenvolvedor na Faixa. Escolha essa opção e clique em OK. O Excel exibe a faixa Desenvolvedor mostrada na Figura 1.1.

Figura 1.1
A Faixa Desenvolvedor fornece uma interface para executar e gravar macros.

O grupo Código da Faixa Desenvolvedor contém ícones utilizados para gravar e reproduzir macros do VBA.

- **O ícone do Visual Basic** — Abre o Visual Basic Editor.
- **O ícone Macros** — Exibe a caixa de diálogo Macro, na qual você pode optar por executar ou editar uma macro a partir da lista de macros.
- O ícone **Gravar Macro** — Inicia o processo de gravação de uma macro.
- O ícone **Usar Referências Relativas** — Alterna entre o uso de gravação relativa ou absoluta. Com a gravação relativa, o Excel gravará que você se moveu três células para baixo. Com a gravação absoluta, o Excel gravará que você selecionou a célula A4.
- **O ícone Segurança de Macro** — Acessa a Central de Confiabilidade, onde você pode escolher ativar ou desativar macros para executar nesse computador.

O grupo Controles da Faixa Desenvolvedor contém um menu Inserir, onde você pode acessar uma variedade de controles de programação que podem ser colocados na planilha. Veja "Atribuindo uma macro a um controle de formulário, uma caixa de texto ou uma forma" mais adiante neste capítulo. Outros ícones nesse grupo permitem trabalhar com os controles na planilha. O botão Executar Caixa de Diálogo permite exibir uma caixa de diálogo personalizada ou formulário do usuário que você projetou no VBA. Para informações adicionais sobre userforms, consulte o Capítulo 10, "Userforms — uma introdução".

> **NOTA** O grupo XML da Faixa Desenvolvedor contém ferramentas para importar e exportar documentos XML. Veja o Capítulo 17, "XML no Excel 2007".

Segurança de macro

Depois que macros VBA foram utilizadas como o método de entrega para alguns vírus ostensivos, a Microsoft mudou as configurações padrão de segurança para impedir a execução dessas macros. Portanto, antes de começarmos a discutir a gravação de uma macro, precisamos mostrar como ajustar as configurações padrão.

NOVO No Excel 2007, você pode ajustar as configurações de segurança globalmente ou controlar as configurações de macro para certas pastas de trabalho, salvando as pastas de trabalho em um local confiável. Qualquer pasta de trabalho armazenada em uma pasta marcada como local confiável terá suas macros ativadas automaticamente.

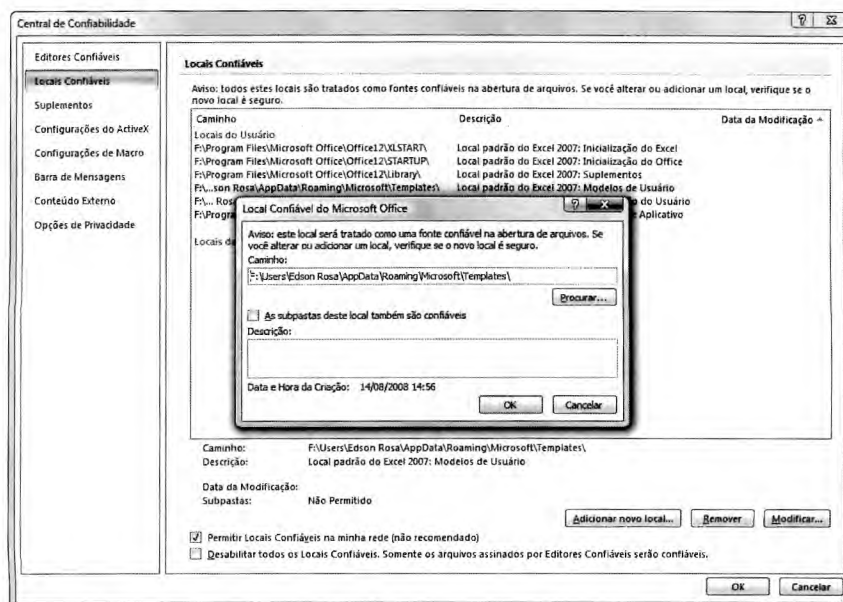
Você pode encontrar as configurações de segurança de macro sob o ícone Segurança de Macro na faixa Desenvolvedor. Ao clicar nesse ícone, você verá a categoria Configurações de Macro da Central de Confiabilidade. Você pode utilizar a barra de navegação esquerda na caixa de diálogo para acessar a lista Locais Confiáveis.

Adicionando um local confiável

NOVO Você pode optar por armazenar as pastas de trabalho de macros em uma pasta marcada como um local confiável. Quaisquer macros em uma pasta de trabalho armazenada em uma pasta confiável serão ativadas. A Microsoft sugere a unidade de disco como um local confiável. A configuração padrão é que não é possível confiar em um local na unidade de rede.

Para especificar um local confiável, siga estes passos:

1. Clique em Segurança de Macro na faixa Desenvolvedor.
2. Clique em Locais Confiáveis no painel esquerdo de navegação da Central de Confiabilidade.
3. Se quiser confiar em uma local da unidade de rede, escolha Permitir Locais Confiáveis na Minha Rede (Não Recomendado).
4. Clique no botão Adicionar Novo Local. O Excel exibe a caixa de diálogo Local Confiável do Microsoft Office (veja Figura 1.2).

Figura 1.2
Gerencie as pastas confiáveis na categoria Locais Confiáveis da Central de Confiabilidade.

5. Clique no botão Procurar. O Excel exibe a caixa de diálogo Procurar.

6. Navegue até a pasta pai da pasta que você quer que especificar como um local confiável. Clique na pasta confiável. Embora o nome da pasta não apareça na caixa Nome da Pasta, você pode clicar em OK. O nome correto da pasta aparecerá no caixa de diálogo Procurar.
7. Se quiser confiar nas subpastas da pasta selecionada, escolha As Subpastas Deste Local Também São Confiáveis.
8. Clique em OK para adicionar a pasta à lista Locais Confiáveis.

Embora locais confiáveis não sejam novos no Excel 2007, a Microsoft tornou mais fácil de encontrar o processo de adição de locais confiáveis no Excel 2007.

ATENÇÃO

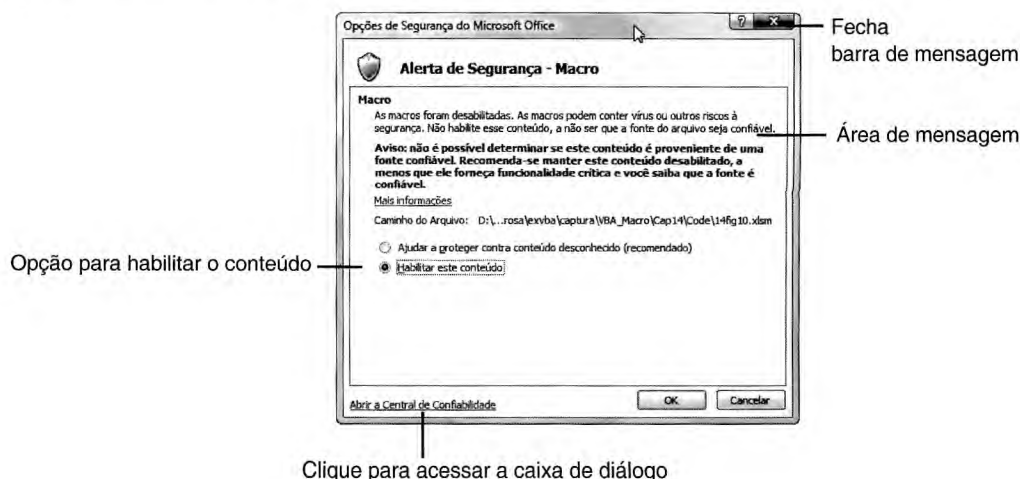
Seja cuidadoso ao selecionar um local confiável. Se você der um clique duplo em um anexo do Excel de um e-mail, o Outlook armazenará temporariamente o arquivo em uma pasta temp na sua unidade C: unidade de rede. Não é recomendável adicionar globalmente C:\ e todas as subpastas à lista Locais Confiáveis.

Utilizando as Configurações de Macro para habilitar macros nas pastas de trabalho fora dos Locais Confiáveis

NOVO Para todas as macros não armazenadas em um local confiável, o Excel conta com as configurações de macro. As configurações Baixa, Média, Alta e Muito Alta utilizadas nos últimos oito anos foram renomeadas.

Para acessar as configurações de macro, clique em Segurança de Macro na faixa Desenvolvedor. O Excel exibe a categoria Configurações de Macro da caixa de diálogo Central de Confiabilidade. Escolha a segunda opção, Desabilitar Todas as Macros com Notificação. A seguir há uma descrição de cada opção:

- **Desabilitar Todas as Macros sem Notificação** — Essa configuração impede que todas as macros sejam executadas. Essa configuração é para usuários que nunca executarão macros. Como no momento você está lendo um livro que ensina a utilizar macros, suponho que essa não seja sua opção. Essa configuração é mais ou menos equivalente à antiga configuração Segurança Muito Alta no Excel 2003. Com essa configuração, apenas as macros nas pastas de locais confiáveis podem ser executadas.
- **Desabilitar Todas as Macros com Notificação** — Essa opção é semelhante à Segurança Média no Excel 2003 e é a que eu recomendo. No Excel 2003, uma configuração Média exibiria uma caixa quando você abrisse um arquivo contendo macros. Essa caixa forçaria o usuário a escolher Habilitar ou Desabilitar. Acho que muitos usuários iniciantes do Excel selecionariam aleatoriamente essa caixa. No Excel 2007, a mensagem é exibida na Área de Mensagem em que as macros foram desativadas. Você pode escolher habilitar o conteúdo clicando nessa opção, como mostrado na Figura 1.3.
- **Desabilitar Todas as Macros, Exceto as Digitalmente Assinadas** — Essa opção exigiria obter uma ferramenta de assinatura digital da VeriSign ou outro provedor. Ela poderia ser apropriada se você vender suplementos a outras pessoas, mas um pouco trabalhosa se você só quiser escrever suas macros para uso próprio.
- **Habilitar Todas as Macros (Não Recomendado: Códigos Potencialmente Perigosos Podem Ser Executados)** — Essa opção é semelhante à Baixa Segurança de Macro no Excel 2003. Embora a trabalheira seja menor, tal opção também deixa o computador vulnerável a ataques de vírus maliciosos do tipo Melissa. A Microsoft sugere não utilizar essa configuração.

Figura 1.3
Abra uma pasta de trabalho de macros utilizando a opção Desabilitar Todas as Macros com Notificação e você poderá ativar facilmente as macros.

Utilizando a opção Desabilitar Todas as Macros com Notificação

Minha recomendação é configurar suas opções de macro como Desativar Todo o Conteúdo com Notificação. Se utilizar essa configuração e abrir uma pasta de trabalho que contenha macros, você verá um Aviso de Segurança na área logo acima da barra de fórmula. Siga estes passos para habilitar macros:

1. Clique no botão Opções ao lado de Aviso de Segurança. O Excel exibe as Opções de Segurança do Microsoft Office.
2. Supondo que você esteja esperando ver macros nessa pasta de trabalho, escolha Habilitar Este Conteúdo.
3. Clique em OK. As macros agora estão habilitadas nessa pasta de trabalho.

Se você não quiser habilitar as macros da pasta de trabalho atual, feche o Alerta de Segurança clicando no X na extremidade direita da barra de mensagem.

Se você esquecer-se de habilitar as macros e tentar executar uma, o Excel indicará que não é possível executá-las porque todas foram desativadas. Se precisar reabrir a barra de mensagens, utilize a caixa de seleção Barra de Mensagens no grupo Mostrar/Ocultar da Faixa Exibição.

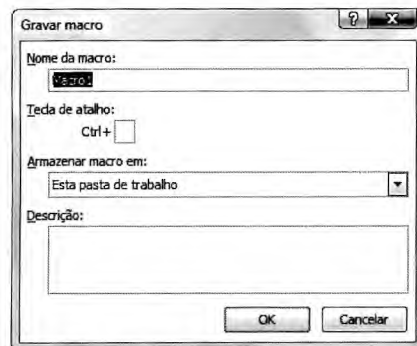
Visão geral da gravação, armazenamento e execução de uma macro

Gravar uma macro é muito útil quando não se tem experiência suficiente para escrever linhas de código em uma macro. À medida que você adquire mais experiência, começará a gravar cada vez menos linhas de código.

Para iniciar a gravação de uma macro, selecione Gravar Macro da Faixa Desenvolvedor. Antes de a gravação começar, o Excel exibe a caixa de diálogo Gravar Macros, conforme mostrado na Figura 1.4.

Preenchendo a caixa de diálogo Gravar Macros

No campo Nome de Macro, digite um nome para a macro. Certifique-se de digitar caracteres contínuos; por exemplo, digite `Macro1` e não Macro 1 (com um espaço). Supondo que em breve você criará muitas macros, utilize um nome descritivo para a macro. Um nome como FormatReport é mais útil do que Macro1.

Figura 1.4
Utilize a caixa de diálogo Gravar Macro para atribuir um nome e uma tecla de atalho à macro prestes a ser gravada.

O segundo campo na caixa de diálogo Gravar Macro é uma tecla de atalho. Se você digitar **J** nesse campo e então pressionar Ctrl+J, essa macro será executada.

Na caixa de diálogo Gravar Macro, você pode escolher onde quer salvar uma macro ao gravá-la: Pasta de Trabalho Macro Pessoais, Nova Pasta de Trabalho, Esta Pasta de Trabalho. É recomendável armazenar macros relacionadas a uma determinada pasta de trabalho em Esta Pasta de Trabalho.

A Pasta de Trabalho Pessoal de Macros (Pessoal.xlsm) não é uma pasta de trabalho aberta; ela é criada se você optar por salvar a gravação na Pasta de Trabalho Pessoal de Macros. Essa pasta de trabalho é utilizada para salvar uma macro em uma pasta de trabalho que abrirá automaticamente quando você iniciar o Excel, permitindo assim o uso da macro. Depois que o Excel é iniciado, a pasta de trabalho permanece oculta. Se quiser exibi-la, selecione Reexibir no menu Janela.

Não é recomendável utilizar a pasta de trabalho pessoal para cada macro salva. Salve apenas aquelas que o ajudarão nas tarefas gerais — não nas tarefas que são realizadas em uma planilha ou pasta de trabalho específica.

A quarta caixa na caixa de diálogo Gravar Macro é utilizada para uma descrição. Ela será adicionada como um comentário ao começo da sua macro. Observe que as versões anteriores do Excel anotavam automaticamente a data e o nome de usuário da pessoa que estivesse gravando a macro. O Excel 2007 não mais insere automaticamente essas informações no campo Descrição.

Depois de selecionar o local em que você quer armazenar a macro, clique em OK. Grave a macro. Ao terminar a gravação, clique no ícone Parar Gravação na Faixa Desenvolvedor.

DICA

Você também pode acessar o ícone Parar Gravação no canto inferior esquerdo da janela do Excel. Procure um pequeno quadrado azul à direita da palavra *Pronto* na barra de status. Utilizar o botão Parar pode ser mais conveniente do que retornar à Faixa Desenvolvedor. Se você não estiver gravando uma macro, esse ícone transforma-se em um pequeno ponto vermelho na planilha do Excel. Esse ícone é um atalho para começar a gravar uma nova macro.

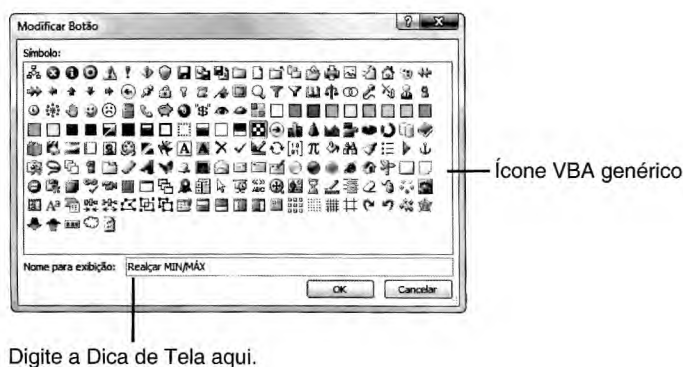
Executando uma macro

Se você atribuiu uma tecla de atalho à macro, pode ativá-la pressionando a combinação de teclas. As macros também podem ser atribuídas a botões da barra de ferramentas, controles de formulários, objetos de desenho ou executadas a partir da barra de ferramentas do Visual Basic.

Criando um botão Macro

Você pode adicionar um ícone à barra de ferramentas Acesso Rápido para executar a macro. Se a macro for armazenada na Pasta de Trabalho Pessoal de Macros, o botão poderá permanecer visível permanentemente na barra de ferramentas Acesso Rápido. Se a macro for armazenada na pasta de trabalho atual, é possível especificar que o ícone só deve aparecer quando a pasta de trabalho for aberta. Siga estes passos para adicionar um botão de macros à barra de ferramentas Acesso Rápido:

1. Clique no botão Office e escolha Opções do Excel para abrir a caixa de diálogo Opções do Excel.
2. Na caixa de diálogo Opções do Excel, escolha a categoria Personalizar à esquerda. (Observe que um atalho para substituir os passos 1 e 2 é clicar com o botão direito do mouse na barra de ferramentas Acesso Rápido e escolher Personalizar Barra de Ferramentas Acesso Rápido.)
3. Se você só for utilizar a macro quando a pasta de trabalho atual estiver aberta, abra a caixa suspensa no canto superior direito e mude de Para Todos os Documentos (Padrão) para Para <NomeDeArquivo.xlsm>. Todos os ícones associados à pasta de trabalho atual são exibidos no final da Barra de Ferramentas Acesso Rápido.
4. Abra a caixa suspensa no canto superior esquerdo e escolha Macros na lista (é a quarta categoria da lista). O Excel exibe uma lista das macros disponíveis na caixa de listagem esquerda.
5. Escolha uma macro na caixa de listagem esquerda. Clique no botão Adicionar na parte central da caixa de diálogo. O Excel move a macro para a caixa de listagem direita. O Excel utiliza um ícone VBA genérico para todas as macros. Você pode mudar o ícone seguindo os passos 6 a 8.
6. Clique na macro na caixa de listagem direita. Clique no botão Modificar, na parte inferior da caixa de listagem direita. O Excel exibe uma lista de 181 possíveis ícones (veja Figura 1.5). Considerando que o Excel 2003 oferecia 4.096 possíveis ícones, mais um editor de ícones, a lista com apenas 181 é uma grande decepção.

Figura 1.5
Anexe uma macro a um botão na barra de ferramentas Acesso Rápido.

7. Escolha um ícone na lista. Na caixa Exibir Nome, substitua o nome da macro por um nome curto que aparecerá na Dica de Tela do ícone.
8. Clique em OK para fechar a caixa de diálogo Modificar Botão.
9. Clique em OK para fechar as Opções do Excel. O novo botão aparece na barra de ferramentas Acesso Rápido.

Atribuindo uma macro a um *controle* de formulário, uma caixa de texto ou uma forma

Se quiser criar uma macro específica para uma pasta de trabalho, armazene a macro na pasta de trabalho e anexe-a a um controle de formulário ou a qualquer objeto na planilha.

Siga estes passos para anexar uma macro a um controle de formulário na planilha:

1. Na Faixa Desenvolvedor, clique no botão Inserir para abrir a lista suspensa. O Excel oferece 12 controles de formulário e 12 controles ActiveX. Muitos ícones têm uma aparência semelhante nessa lista suspensa. Clique no ícone Botão no canto superior esquerdo da lista suspensa Controles de Formulário.

2. Mova o cursor do mouse sobre a planilha; o cursor transforma-se em um sinal de adição.

3. Desenhe um botão na planilha, clicando e mantendo o botão esquerdo do mouse pressionado ao desenhar uma forma de caixa. Libere o botão quando terminar.

4. Selecione uma macro na caixa de diálogo Atribuir Macro e clique em OK. O botão é criado com texto genérico como Botão 1. Para personalizar o texto ou a aparência do botão, siga os passos 5 a 7.

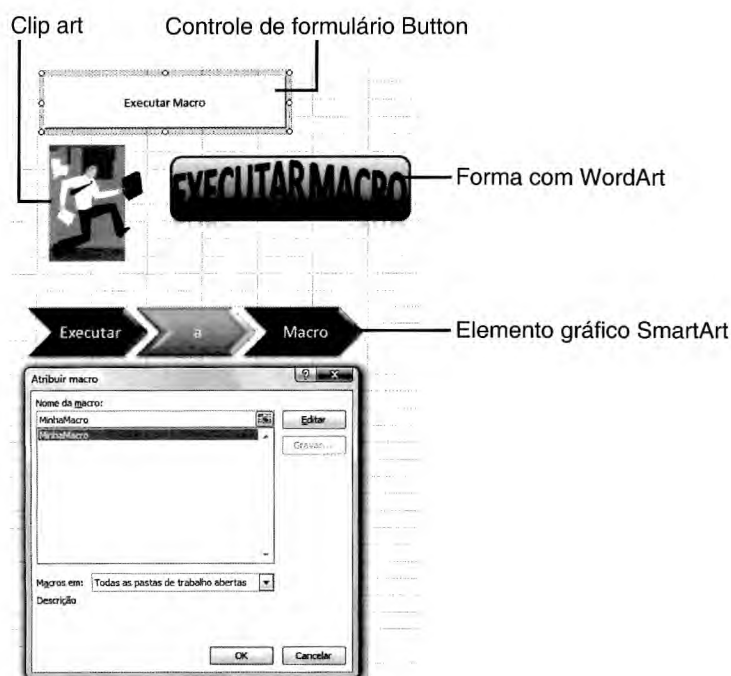
5. Com a tecla Ctrl pressionada, clique no botão para selecioná-lo. Arraste o cursor sobre o texto no botão para selecionar o texto. Digite um novo rótulo para o botão. Observe que, enquanto você digita, a borda de seleção em volta do botão muda de pontos para linhas diagonais a fim de indicar que você está no modo Edição de Texto. Não é possível alterar a cor do botão no modo Edição de Texto. Para sair do modo Edição de Texto, clique nas linhas diagonais para transformá-las em pontos ou, com Ctrl pressionada, clique no botão novamente.

6. Clique com o botão direito do mouse nos pontos que cercam o botão e escolha Formatar Controle. O Excel exibe a caixa de diálogo Formatar Controle com sete guias ao longo da parte superior. Se a caixa de diálogo Formatar Controle tiver apenas uma guia Fonte, você não conseguiu sair do modo Edição de Texto. Feche a caixa de diálogo, com Ctrl pressionada, clique no botão e repita esse passo.

7. Utilize as configurações na caixa de diálogo Formatar Controle para alterar o tamanho da fonte, a cor da fonte, as margens e configurações semelhantes para o controle. Clique em OK para fechar a caixa de diálogo Formatar Controle quando terminar.

8. Clique no botão para executar a macro.

As macros podem ser atribuídas a qualquer objeto de planilha como clip art, uma forma, elementos gráficos SmartArt ou uma caixa de texto. Na Figura 1.6, o botão na parte superior é um controle tradicional de formulário de botão. As outras imagens são clip art, uma forma com WordArt e um elemento gráfico SmartArt. Para atribuir uma macro a qualquer objeto, clique com o botão direito do mouse no objeto e escolha Atribuir Macro.

Figura 1.6
Atribuindo uma macro a um controle de formulário ou a um objeto adequado para macros armazenadas na mesma pasta de trabalho do controle. Você pode atribuir uma macro a qualquer um desses objetos.

Utilizando novos tipos de arquivo no Excel 2007

O Excel 2007 oferece suporte a quatro tipos de arquivo. Não é permitido armazenar macros no tipo de arquivo padrão. Você tem de utilizar a opção Salvar Como para todas as pastas de trabalho de macros, ou pode mudar o tipo de arquivo padrão utilizado pelo Excel 2007.

Os tipos de arquivo disponíveis são mostrados a seguir:

- **Pasta de trabalho do Excel (.xlsx)** — Os arquivos são armazenados como uma série de objetos XML e então compactados em um único arquivo. Esse novo paradigma de salvar arquivos no Excel 2007 permite tamanhos de arquivo significativamente menores. Ele também permite que outros aplicativos (mesmo o Bloco de Notas!) editem ou criem pastas de trabalho do Excel. Infelizmente, as macros não podem ser armazenadas em arquivos com uma extensão .xlsx.
- **Pasta de trabalho do Excel com macros ativadas (.xlsm)** — Esse formato é semelhante ao formato padrão .xlsx, exceto que as macros permanecem ativadas. O conceito básico é que, se houver um arquivo .xlsx, você não precisará se preocupar com macros maliciosas, mas se houver um arquivo .xlsm, haverá grandes probabilidades de que existam macros anexadas.
- **Pasta de trabalho binária do Excel (.xlsb)** — Esse é um formato binário projetado para tratar o maior tamanho de grade de 1,1 milhão de linhas no Excel 2007. Todas as versões prévias do Excel armazenavam os arquivos em um formato binário proprietário. Embora formatos binários possam ser carregados mais rapidamente, eles têm maior propensão a se corromperem; alguns bits perdidos podem destruir todo o arquivo. Não é permitido usar macros nesse formato.
- **Pasta de trabalho do Excel 97-2003 (.xls)** — Esse formato produz arquivos que podem ser lidos por qualquer pessoa a partir de versões legadas do Excel. Ele é um formato binário e as macros podem ser ativadas. Entretanto, ao salvar nesse formato, você perde acesso a qualquer célula fora de A1:IV65536. Você perde acesso a todos os novos recursos no Excel.

Para evitar o trabalho de escolher uma pasta de trabalho com macros ativadas na caixa de diálogo Salvar Como, personalize sua cópia do Excel para que você sempre salve os novos arquivos no formato .xlsm. Siga estes passos:

1. Clique no botão Office e escolha Opções do Excel.
2. Na caixa de diálogo Opções do Excel, escolha a categoria Salvar no painel de navegação esquerdo.
3. A primeira lista suspensa é Salvar Arquivos Neste Formato. Abra-a e escolha Pasta de Trabalho Habilitada para Macro do Excel (*.xlsm). Clique em OK.

NOTA

Embora você e eu não tenhamos medo de utilizar macros, encontrei algumas pessoas que pareciam enlouquecer completamente quando viam o tipo de arquivo .xlsm. Elas pareciam realmente furiosas quando eu enviava um arquivo .xlsm que não tinha nenhuma macro. A reação dessas pessoas lembrava uma frase do Rei Arthur no filme *Monty Python e o Cálice Sagrado:* "Você me deixa muito perturbado!"

Se encontrar alguém que parece ter um medo irracional em relação ao tipo de arquivo .xlsm, lembre-o destes pontos:

- Todas as pastas de trabalho criadas nos últimos 20 anos poderiam conter macros e a maioria não continha.
- Se alguém estiver tentando evitar macros, deverá utilizar as configurações de segurança para impedir que as macros sejam executadas (consulte novamente a Figura 1.3). Ainda poderá abrir o arquivo .xlsm para obter os dados na planilha.

Com esses argumentos, esperançosamente você irá superar qualquer medo irracional em relação ao tipo de arquivo .xlsm e torná-lo seu tipo de arquivo padrão.

Entendendo o Editor do Visual Basic

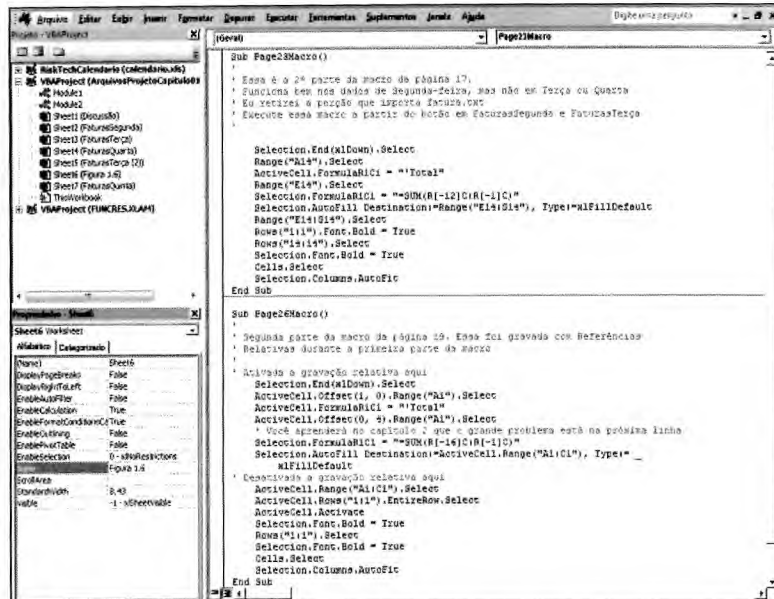
A Figura 1.7 mostra um exemplo da tela típica do Editor do VB. Você pode ver três janelas: Project Explorer, Propriedades e a janela de Programação. Não se preocupe se sua janela não for exatamente como essa; à medida que revisamos o editor, mostraremos como exibir as janelas de que você precisa.

Configurações do Editor do VB

Várias configurações do Editor do VB permitem que você o personalize da maneira como preferir. Revisaremos apenas aquelas que o ajudarão na sua programação.

Personalizando configurações de opções do Editor do VB

Em Ferramentas, Opções, Editor, você encontrará várias configurações úteis. Todas as configurações, exceto uma, são corretamente definidas por padrão. Essa configuração, que exige que você faça algumas considerações, é Requerer Declaração de Variável. Por

Figura 1.7
A janela do Editor do VB.

padrão, o Excel não exige que você declare variáveis. Prefiro essa configuração, pois você pode economizar tempo criando um programa. Minha co-autora, Tracy, prefere alterar essa opção para exigir a declaração de variável. Isso força o compilador a parar se encontrar uma variável que ele não reconhece e reduz a digitação incorreta de nomes de variáveis. É uma questão de preferência pessoal habilitar ou desabilitar essa opção.

Habilitando assinaturas digitais

Se for como eu, você escreve uma grande quantidade de macros pessoais. Depois de algum tempo, torna-se cansativo habilitar as macros criadas. É aqui que entram as assinaturas digitais. É possível comprar uma assinatura digital a partir de vários fornecedores independentes. A Microsoft fornece uma lista de fornecedores aprovados em http://msdn2.microsoft.com/en-us/library/ms995347.aspx. Depois que você adquiriu uma assinatura digital, use Ferramentas, Assinatura Digital, para anexar a assinatura ao seu projeto VBA.

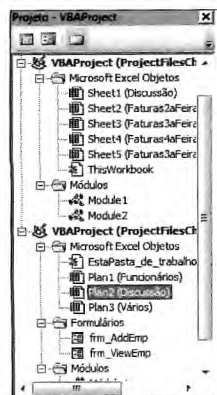
O Project Explorer

O Project Explorer relaciona todas as pastas de trabalho abertas e suplementos que são carregados. Se clicar no ícone + ao lado do Projeto do VBA, você verá que há uma pasta com objetos do Microsoft Excel. Também poderá haver pastas para formulários, módulos de classe e módulos (padrão). Todas as pastas incluem um ou mais componentes individuais.

Clicar com o botão direito do mouse em um componente e selecionar Exibir, Código, ou apenas dar um clique duplo nos componentes, exibe qualquer código na janela Programação (exceto para userforms, onde dar um clique duplo exibe o userform na visualização Design).

Para exibir essa janela, selecione Exibir, Project Explorer a partir do menu, pressione Ctrl+R ou clique no ícone Project Explorer na barra de ferramentas.

A Figura 1.8 mostra o painel Project Explorer. Esse painel pode mostrar objetos do Microsoft Excel, userforms, módulos e módulos de classe.

Figura 1.8
O Project Explorer, exibindo diferentes tipos de módulos.

Para inserir um módulo, clique com o botão direito do mouse no projeto, selecione Inserir e, então, selecione o tipo de módulo que você quer. Os módulos disponíveis são os seguintes:

- **Objetos do Microsoft Excel** — Por padrão, um projeto consiste em módulos de planilha para cada planilha na pasta de trabalho e em um único módulo EstaPasta_de_trabalho. O código específico a uma planilha, como controles ou eventos de planilha, é posicionado na planilha correspondente. Os eventos de pasta de trabalho são colocados no módulo EstaPasta_de_trabalho. Você aprenderá mais sobre eventos no Capítulo 9, "Programação de eventos".
- **Formulários** — O Excel permite que você crie seus próprios formulários para interagir com o usuário. Você aprenderá mais sobre esses formulários no Capítulo 10, "Userforms — uma introdução".
- **Módulos** — Ao gravar uma macro, o Excel cria automaticamente um módulo para colocar o código. É nesses tipos de módulos que a maior parte do seu código residirá.
- **Módulos de classe** — Módulos de classe são a maneira como o Excel permite que você crie seus próprios objetos. Além disso, os módulos de classe permitem que fragmentos do código sejam compartilhados entre programadores sem que eles precisem entender como isso funciona. Você aprenderá mais sobre os módulos de classe no Capítulo 22, "Criando classes, registros e coleções".

A janela Propriedades

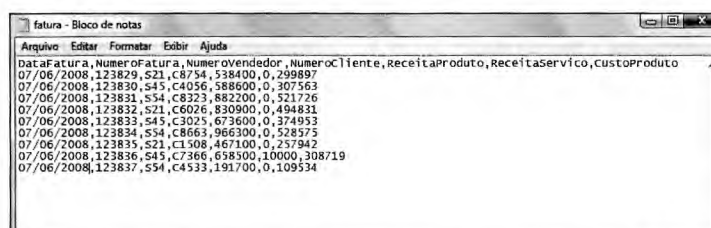
A janela Propriedades permite editar as propriedades de vários componentes — planilhas, pastas de trabalho, módulos e controles de formulário. A lista Propriedades varia de acordo com o componente selecionado.

Para exibir essa janela, selecione Exibir, Propriedades no menu, pressione F4 ou clique no ícone Propriedades de Projeto na barra de ferramentas.

Entendendo as falhas do programa de gravação de macros

Suponha que você trabalhe em um departamento de contabilidade. Todos os dias você recebe um arquivo de texto a partir do sistema da empresa que mostra todas as faturas geradas no dia anterior. Esse arquivo de texto contém vírgulas que separam cada campo. As colunas no arquivo são DataFatura, NumeroFatura, NumeroVendedor, NumeroCliente, ReceitaProduto, ReceitaServico e CustoProduto (veja Figura 1.9).

fatura - Bloco de notas
Arquivo Editar Formatar Exibir Ajuda
```
DataFatura,NumeroFatura,NumeroVendedor,NumeroCliente,ReceitaProduto,ReceitaServico,CustoProduto
07/06/2008,123829,S21,C8754,538400,0,299897
07/06/2008,123830,S45,C4056,588600,0,307563
07/06/2008,123831,S54,C8323,882200,0,521726
07/06/2008,123832,S21,C6026,830900,0,494831
07/06/2008,123833,S45,C3025,673600,0,374953
07/06/2008,123834,S54,C8663,966300,0,528575
07/06/2008,123835,S21,C1508,467100,0,257942
07/06/2008,123836,S45,C7366,658500,10000,308719
07/06/2008,123837,S54,C4533,191700,0,109534
```

Figura 1.9
Arquivo fatura.txt

Ao chegar ao trabalho pela manhã, você importa esse arquivo manualmente para o Excel, adiciona uma linha total aos dados, aplica negrito aos títulos e imprime o relatório para alguns gerentes.

Isso parece um processo relativamente simples que, de modo ideal, seria adequado ao uso do programa de gravação de macros. Mas, devido a alguns problemas com o programa de gravação de macros, suas poucas primeiras tentativas podem não ser bem-sucedidas.

Estudo de caso

Preparando para gravar a macro

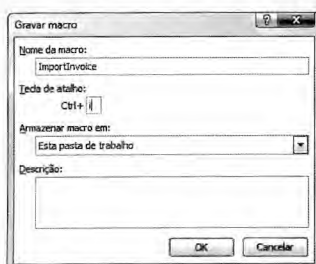
Essa é uma tarefa perfeita para uma macro. Antes de gravar alguma macro, pense nos passos que você seguirá antes de começar. No nosso caso, esses passos são:

1. Clique no botão Office e selecione Abrir.
2. Navegue até a pasta em que fatura.txt está armazenada.
3. Escolha Todos os Arquivos (*.*) na lista suspensa Arquivos de Tipo.

4. Selecione fatura.txt.
5. Clique em Abrir.
6. No Assistente de Importação de Texto — Passo 1 de 3, selecione Delimitado na seção Tipo de Dados Originais.
7. Clique em Avançar.
8. No Assistente de Importação de Texto — Passo 2 de 3, desmarque a caixa de seleção Tabulação e marque Vírgula na seção Delimitadores.
9. Clique em Avançar.
10. No Assistente de Importação de Texto — Passo 3 de 3, selecione Geral, na seção Formato dos Dados da Coluna, e mude-o para `Data:ADM`.
11. Clique em Concluir para importar o arquivo.
12. Pressione a tecla End seguida da tecla de seta para baixo para mover-se até a última linha de dados.
13. Pressione a seta para baixo mais uma vez para mover-se até à linha total.
14. Digite a palavra `Total`.
15. Pressione a tecla de seta para a direita quatro vezes para mover-se até a coluna E da linha total.
16. Clique no botão AutoSoma e pressione Ctrl+Enter para adicionar um total à coluna (Receita do Produto) e, ao mesmo tempo, permanecer nessa célula.
17. Arraste a alça Autopreenchimento e arraste-a da coluna E até a coluna G para copiar a fórmula total para as colunas F e G.
18. Destaque a Linha 1 e clique no ícone Negrito, na faixa Início, para definir os títulos em negrito.
19. Destaque a linha Total e clique no ícone Negrito, na faixa Início, para aplicar negrito aos totais.
20. Pressione Ctrl+T para selecionar todas as células.
21. Na Faixa Início, selecione Formatar, AutoAjuste Largura da Coluna.

Depois de memorizar esses passos, você está pronto para gravar sua primeira macro. Abra uma pasta de trabalho vazia e salve-a com um nome como MacroParaImportarFaturas.xlsm. Clique no botão Gravar Macro na faixa Desenvolvedor.

Na caixa de diálogo Gravar Macro, o nome padrão da macro é Macro1. Mude isso para algo descritivo como ImportarFatura. Certifique-se de que as macros serão armazenadas em Esta Pasta de Trabalho. Você poderá querer uma maneira fácil de executar essa macro mais tarde; então, insira a letra `i` no campo Tecla de Atalho. No campo Descrição, adicione um pequeno texto descritivo para informar o que a macro faz (veja Figura 1.10). Quando tiver terminado, dê um clique em OK.

Figura 1.10
Antes de gravar a macro, complete a caixa de diálogo Gravar Macro.

Gravando a macro

Não fique tenso, agora o Programa de Gravação de Macros vai gravar todos os seus movimentos. Seu objetivo é tentar realizar os passos na ordem exata, sem ações irrelevantes. Se você se mover acidentalmente para a coluna F e voltar à coluna E para inserir o primeiro total, a macro gravada cometerá o mesmo erro continuamente por dias. Macros gravadas são muito rápidas, mas isso não é nada comparado ao fato de que você verá seus erros sendo cometidos repetidamente pelo programa de gravação de macros.

Preste atenção ao executar todas as ações necessárias para criar o relatório. Depois de completar o passo final, clique no botão Parar no canto inferior esquerdo da janela Excel, ou clique em Parar de Gravar, na faixa Desenvolvedor.

Está na hora de examinar seu código. Mude para o Editor do VB, escolhendo Visual Basic na faixa Desenvolvedor ou pressionando Alt+F11.

Examinando o código na janela Programação

Vejamos o código que você acabou de gravar no estudo de caso; não se preocupe se ele ainda não fizer sentido.

Para abrir o Editor do VB, pressione Alt+F11. Em seu Projeto de VBA (ImportarFatura.xls), localize o componente Module1, clique nele com o botão direito do mouse e selecione Exibir Código. Note que algumas linhas iniciam com um apóstrofo — elas são comentários e são ignoradas pelo programa. O programa de gravação de macros inicia as macros com alguns comentários, utilizando a descrição que você inseriu na caixa de diálogo Gravar Macro. O comentário para Tecla de Atalho está aí para lembrá-lo do atalho.

> **NOTA** O comentário *não* atribui o atalho. Se você mudar o comentário para Ctrl+J, isso não mudará o atalho. Você deve alterar a configuração na caixa de diálogo Macro no Excel ou executar esta linha de código:

```
Application.MacroOptions Macro:="ImportInvoice", _
    Description:="", ShortcutKey:="j"
    ActiveCell.FormulaR1C1 = "Hello World!"
    Range("A3").Select
```

Em geral, o código da macro gravada macro é muito organizado (veja Figura 1.11). Toda linha de não-comentário do código apresenta um recuo de quatro caracteres. Se uma linha tiver mais de 100 caracteres, o gravador a dividirá em várias linhas e as recuará com quatro caracteres adicionais. Para continuar uma linha de código, digite um espaço e um sublinhado no final da linha. Observe que as limitações físicas deste livro não permitem 100 caracteres em uma única linha. Dividirei as linhas quando elas alcançarem 80 caracteres para que possam se ajustar nesta página. Sua macro gravada pode ser ligeiramente diferente daquelas que aparecem aqui.

Figura 1.11
A macro gravada tem uma ótima aparência e um recuo bem diagramado.

```
Sub ImportInvoice()
'
' ImportInvoice Macro
'
'
' Atalho de teclado: Ctrl+i
'
    Workbooks.OpenText Filename:= _
        "fatura.txt", Origin _
        :=437, StartRow:=1, DataType:=xlDelimited, TextQualifier:=xlDoubleQuote _
        , ConsecutiveDelimiter:=False, Tab:=True, Semicolon:=False, Comma:=True _
        , Space:=False, Other:=False, FieldInfo:=Array(Array(1, 3), Array(2, 1), _
        Array(3, 1), Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)), TrailingMinusNumbers
        :=True
    Selection.End(xlDown).Select
    Range("A14").Select
    ActiveCell.FormulaR1C1 = "'Total"
    Range("E14").Select
    Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
    Selection.AutoFill Destination:=Range("E14:G14"), Type:=xlFillDefault
    Range("E14:G14").Select
    Rows("1:1").Font.Bold = True
    Rows("14:14").Select
    Selection.Font.Bold = True
    Cells.Select
    Selection.Columns.AutoFit
End Sub
```

Considere que as próximas sete linhas do código gravado são, na verdade, apenas uma linha de código dividida em sete linhas, por uma questão de legibilidade:

```
Workbooks.OpenText Filename:= _
    "C:\fatura.txt", Origin:=437, StartRow:=1, DataType:=xlDelimited, _
    TextQualifier:=xlDoubleQuote, ConsecutiveDelimiter:=False, _
    Tab:=True, Semicolon:=False, Comma:=True, Space:=False, _
    Other:=False, FieldInfo:=Array(Array(1, 3), Array(2, 1), Array(3, 1), _
    Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)), _
   TrailingMinusNumbers:=True
```

Contando aquela acima como uma única linha, o programa de gravação de macros foi capaz de gravar nosso processo de 21 passos em 14 linhas de código, o que é muito impressionante.

> **NOTA** Cada ação que você realiza na interface do usuário do Excel pode ser equivalente a uma ou mais linhas do código gravado. Algumas ações podem gerar várias linhas de código.

Sempre é uma boa idéia testar a macro. Retorne à interface normal do Excel pressionando Alt+F11. Feche fatura.txt sem salvar as alterações. ImportarFatura.xls ainda está aberta.

Pressione Ctrl+I para executar a *macro* gravada. Ela funciona primorosamente! Os dados são importados, os totais são calculados, a formatação de negrito é aplicada e as colunas têm uma largura maior. Parece uma solução perfeita (veja *Figura 1.12)*.

Figura 1.12
A macro formata muito bem os dados da planilha.

	A	B	C	D	E	F	G
1	DataFatura	NumeroFatura	NumeroVendedor	NumeroCliente	ReceitaProduto	ReceitaServico	CustoProduto
2	07/06/2008	123801	S82	C8754	639600	12000	325438
3	07/06/2008	123802	S93	C7874	964600	0	435587
4	07/06/2008	123803	S43	C4844	988900	0	587630
5	07/06/2008	123804	S54	C4940	673800	15000	346164
6	07/06/2008	123805	S43	C7969	513500	0	233842
7	07/06/2008	123806	S93	C8468	760600	0	355305
8	07/06/2008	123807	S82	C1620	894100	0	457577
9	07/06/2008	123808	S17	C3238	316200	45000	161877
10	07/06/2008	123809	S32	C5214	111500	0	62956
11	07/06/2008	123810	S45	C3717	747600	0	444162
12	07/06/2008	123811	S87	C7492	857400	0	410493
13	07/06/2008	123812	S43	C7780	200700	0	97937
14	**Total**				**766850**	**72000**	**3918968**
15							

Executar a mesma macro em outro dia produz resultados indesejáveis

Assim, você salvou o arquivo de macro. No dia seguinte, você chega ao trabalho e há um novo arquivo fatura.txt no sistema. Você abre a macro, pressiona Ctrl+I para executá-la e tem uma surpresa desagradável. O fato é que os dados de 7 de junho apresentam 12 faturas e os de 8 de junho, 16 faturas. Mas a macro gravada adicionou cegamente os totais na Linha 14, porque foi nessa linha que colocamos os totais quando a macro foi gravada (veja Figura 1.13).

Figura 1.13
No outro dia, a macro falha terrivelmente. A macro gravada deveria adicionar um total ao final dos dados, mas o gravador fez uma macro adicionar sempre os totais à linha 14.

	A	B	C	D	E	F	G
1	DataFatura	NumeroFatura	NumeroVendedor	NumeroCliente	ReceitaProduto	ReceitaServico	CustoProduto
2	08/06/2008	123813	S82	C8754	716100	12000	423986
3	08/06/2008	123814		C4894	224200	0	131243
4	08/06/2008	123815	S43	C7278	277000	0	139208
5	08/06/2008	123816	S54	C6425	746100	15000	350683
6	08/06/2008	123817	S43	C6291	928300	0	488988
7	08/06/2008	123818	S43	C1000	723200	0	383069
8	08/06/2008	123819	S82	C6025	982600	0	544025
9	08/06/2008	123820	S17	C8026	490100	45000	243808
10	08/06/2008	123821	S43	C4244	615800	0	300579
11	08/06/2008	123822	S45	C1007	271300	0	153253
12	08/06/2008	123823	S87	C1878	338100	0	165666
13	08/06/2008	123824	S43	C3068	567900	0	265775
14	**Total**	**123825**	**S87**	**C4913**	**6880700**	**72000**	**3918968**
15	08/06/2008	123826	S55	C7181	37900	0	19811
16	08/06/2008	123827	S43	C7570	582700	0	292000
17	08/06/2008	123828	S87	C5302	495000	0	241504
18							

Esse primeiro problema surge porque o programa de gravação de macros grava por padrão todas as suas ações no modo absoluto. Em vez de utilizar o estado padrão do programa de gravação de macros, a próxima seção discute a gravação relativa e como isso pode fazer você chegar mais perto da solução final.

Uma possível solução: utilizar referências relativas ao gravar

Por padrão, o programa de gravação de macros grava todas as ações como *absolutas*. Se você navegar até a Linha 14 ao gravar a macro segunda-feira, a macro sempre será executada na Linha 14. Ao lidar com números variáveis de linhas de dados, isso raramente será adequado.

Há uma opção ao uso de referências relativas durante a gravação.

Macros gravadas com referências absolutas notam o endereço real do ponteiro da célula (por exemplo, A14). As macros gravadas com referências relativas notam que o ponteiro de célula deve se mover certo número de linhas e colunas a partir da sua posição atual. Por exemplo, se o ponteiro da célula iniciar na célula A1, o código `ActiveCell.Offset(16, 1).Select` moverá o ponteiro da célula até B17 (a célula 16 linhas abaixo e uma coluna à direita).

Vamos testar o mesmo estudo de caso mais uma vez, agora utilizando referências relativas. A solução está prestes a funcionar.

Estudo de caso

Vamos tentar gravar a macro novamente, utilizando referências relativas. Feche `fatura.txt` sem salvar as alterações. Na pasta de trabalho ImportarFatura.xls, grave uma nova macro escolhendo Gravar Macro na faixa Desenvolvedor. Atribua `ImportInvoicesRelative` como o nome à nova macro e também uma tecla de atalho diferente, *como Ctrl+J (veja* Figura 1.14).

Figura 1.14
Preparando-se para uma segunda tentativa.

À medida que começa a gravar a macro, analise o processo de abrir o arquivo `fatura.txt`. Antes de navegar até a última linha de dados (pressione a tecla End+tecla de seta para baixo), clique no botão Usar Referências Relativas na faixa Desenvolvedor (veja Figura 1.2).

Prossiga com as ações no script do estudo de caso:

1. Pressione a tecla End seguida da tecla de seta para baixo para mover-se até a última linha de dados.
2. Pressione a seta para baixo mais uma vez para mover-se até à linha total.
3. Digite a palavra `Total`.
4. Pressione a tecla de seta para a direita quatro vezes para mover-se até a coluna E da linha Total.
5. Pressione o botão AutoSoma e, então, pressione Ctrl+Enter para adicionar um total à coluna ReceitaProduto enquanto permanece nessa célula.
6. Selecione a alça de preenchimento automático e arraste da coluna E à coluna G a fim de copiar a fórmula total para as colunas F e G.
7. Pressione Shift+barra de espaço para selecionar a linha inteira e aplicar a formatação de negrito.

Nesse ponto, você precisa mover-se até a Célula A1 a fim de aplicar negrito aos títulos. Não é recomendável que o programa de gravação de macros grave o movimento entre a Linha 18 e a Linha 1 — esse movimento seria gravado como 17 linhas para cima, o que poderia não ser correto amanhã. Antes de mover-se para A1, desabilite o botão Usar Gravação Relativa e, então, continue gravando o resto da macro.

8. Destaque a Linha 1 e clique no ícone Negrito para definir os títulos em negrito.
9. Pressione Ctrl+T para selecionar todas as células.
10. Na Faixa Início, selecione Formatar, AutoAjuste da Largura da Coluna.
11. Pare a gravação.

Pressione Alt+F11 para acessar o Editor do VB a fim de revisar o código. A nova macro aparece no Module1 abaixo da macro anterior.

ATENÇÃO

Se você fechar o Excel entre a gravação da primeira e da segunda macro, o Excel irá inserir um novo módulo, chamado Module2, para a macro recém-gravada.

Editei o código a seguir com dois comentários para mostrar onde me lembro de ativar e desativar a gravação relativa.

```
Sub ImportInvoicesRelative
'
'Isso foi gravado com gravação relativa
'As referências habilitadas durante a primeira parte da macro
'
'Habilitei a gravação relativa aqui
    Selection.End(xlDown).Select
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "'Total"
    ActiveCell.Offset(0, 4).Range("A1").Select
    Selection.FormulaR1C1 = "=SUM(R[-16]C:R[-1]C)"
    Selection.AutoFill Destination:=ActiveCell.Range("A1:C1"), Type:= _
        xlFillDefault
'Desabilitei a gravação relativa aqui
    ActiveCell.Range("A1:C1").Select
    ActiveCell.Rows("1:1").EntireRow.Select
    ActiveCell.Activate
    Selection.Font.Bold = True
    Rows("1:1").Select
```

```
    Selection.Font.Bold = True
    Cells.Select
    Selection.Columns.AutoFit
End Sub
```

Para testar a macro, feche fatura.txt sem salvar e então execute a macro com Ctrl+J. Tudo parece certo — os mesmos resultados são obtidos.

O próximo teste é verificar se o programa funciona no dia seguinte, quando você talvez tenha mais linhas. A Figura 1.15 mostra os dados de 9 de junho.

Figura 1.15
A macro com referências relativas

fatura - Bloco de notas

Arquivo Editar Formatar Exibir Ajuda

```
DataFatura,NumeroFatura,NumeroVendedor,NumeroCliente,ReceitaProduto,ReceitaServico,CustoProduto
9/6/2008,123829,S21,C8754,21000,0,9875
9/6/2008,123830,S45,C3390,188100,0,85083
9/6/2008,123831,S54,C2523,510600,0,281158
9/6/2008,123832,S21,C5519,86200,0,49967
9/6/2008,123833,S45,C3245,800100,0,388277
9/6/2008,123834,S54,C7796,339000,0,195298
9/6/2008,123835,S21,C1654,161000,0,90761
9/6/2008,123836,S45,C6460,275500,10000,146341
9/6/2008,123837,S54,C5143,925400,0,473515
9/6/2008,123838,S21,C7868,148200,0,75700
9/6/2008,123839,S45,C3310,890200,0,468333
9/6/2008,123840,S54,C2959,986000,0,528980
9/6/2008,123841,S21,C8361,94400,0,53180
9/6/2008,123842,S45,C1842,36500,55000,20696
9/6/2008,123843,S54,C4107,599700,0,276718
9/6/2008,123844,S21,C5205,244900,0,143393
9/6/2008,123845,S45,C7745,63000,0,35102
9/6/2008,123846,S54,C1730,212600,0,117787
9/6/2008,123847,S21,C6292,974700,0,478731
9/6/2008,123848,S45,C2008,327700,0,170968
9/6/2008,123849,S54,C4096,30700,0,18056
```

Abro a pasta `ImportarFatura.xls` e executo a nova macro com Ctrl+J. Dessa vez, tudo parece muito bom. Os totais estão onde deveriam estar. Examine a Figura 1.16 — você vê alguma coisa fora do comum?

Figura 1.16
O resultado da execução da Macro Relativa.

E23 =SOMA(E7:E22)

	A	B	C	D	E	F	G
1	DataFatura	NumeroFatura	NumeroVendedor	NumeroCliente	ReceitaProduto	ReceitaServico	CustoProduto
2	09/06/2008	123829	S21	C8754	21000	0	9875
3	09/06/2008	123830	S45	C3390	188100	0	85083
4	09/06/2008	123831	S54	C2523	510600	0	281158
5	09/06/2008	123832	S21	C5519	86200	0	49967
6	09/06/2008	123833	S45	C3245	800100	0	388277
7	09/06/2008	123834	S54	C7796	339000	0	195298
8	09/06/2008	123835	S21	C1654	161000	0	90761
9	09/06/2008	123836	S45	C6460	275500	10000	146341
10	09/06/2008	123837	S54	C5143	925400	0	473515
11	09/06/2008	123838	S21	C7868	148200	0	75700
12	09/06/2008	123839	S45	C3310	890200	0	468333
13	09/06/2008	123840	S54	C2959	986000	0	528980
14	09/06/2008	123841	S21	C8361	94400	0	53180
15	09/06/2008	123842	S45	C1842	36500	55000	20696
16	09/06/2008	123843	S54	C4107	599700	0	276718
17	09/06/2008	123844	S21	C5205	244900	0	143393
18	09/06/2008	123845	S45	C7745	63000	0	35102
19	09/06/2008	123846	S54	C1730	212600	0	117787
20	09/06/2008	123847	S21	C6292	974700	0	478731
21	09/06/2008	123848	S45	C2008	327700	0	170968
22	09/06/2008	123849	S54	C4096	30700	0	18056
23	Total				6309500	65000	3293559

A fórmula nesta célula refere-se a um intervalo que tem números adjacentes a ele.

Se não for cuidadoso, talvez o seu gerente simplesmente receba esses resultados. E isso seria problemático. Examine a célula E23. Felizmente, o Excel tem um triângulo verde nesse local alertando-o para examinar a célula. Se por acaso você tivesse tentado isso no Excel 95 ou 97 antes que existissem as marcas inteligentes, não haveria nenhum indicador de que algo estava errado.

Mova o ponteiro de célula para E23. Um indicador de alerta abre perto da célula e informa que a fórmula não consegue incluir as células adjacentes. Se examinar a barra de fórmula, você poderá ver que a macro somou apenas as Linhas 7 a 22. Nem a gravação relativa nem a gravação não-relativa é inteligente o bastante para reproduzir a lógica do botão AutoSoma.

Nesse ponto, qualquer pessoa sensata desistiria. Mas imagine que nesse dia em particular poderia haver um número menor de gravações de faturas. O Excel forneceria a fórmula ilógica de `=SOMA(E10:E1048571)` e uma referência circular, como mostrado na Figura 1.17.

Figura 1.17
O resultado da execução da Macro Relativa com um número menor de gravações de fatura.

E11 =SOMA(E10:E1048571)

	A	B	C	D	E	F	G
1	DataFatura	NumeroFatura	NumeroVendedor	NumeroCliente	ReceitaProduto	ReceitaServico	CustoProduto
2	09/06/2008	123829	S21	C8754	21000	0	9875
3	09/06/2008	123830	S45	C3390	188100	0	85083
4	09/06/2008	123831	S54	C2523	510600	0	281158
5	09/06/2008	123832	S21	C5519	86200	0	49967
6	09/06/2008	123833	S45	C3245	800100	0	388277
7	09/06/2008	123834	S54	C7796	339000	0	195298
8	09/06/2008	123835	S21	C1654	161000	0	90761
9	09/06/2008	123836	S45	C6460	275500	10000	146341
10	09/06/2008	123837	S54	C5143	925400	0	473515
11	Total				0	0	0

Imagino que, se você tentou utilizar o programa de gravação de macros, deparou-se com problemas semelhantes àqueles produzidos nos dois últimos estudos de caso. Embora isso seja frustrante, você deve estar ficar feliz em saber que, na verdade, o programa de gravação de macros fez 95% do trabalho relacionado a uma macro útil.

Seu trabalho é identificar onde o programa de gravação de macros provavelmente falhará e, então, analisar o código VBA para corrigir uma ou duas linhas que precisem ser ajustadas a fim de ter uma macro perfeita.

Você precisa examinar e compreender o Visual Basic. Com um pouco de inteligência adicional, você pode produzir macros impressionantes para agilizar seu trabalho diário.

Se for como eu, você está amaldiçoando a Microsoft. Durante alguns dias, desperdiçamos muito tempo e nenhuma macro funcionou. O que torna as coisas piores é que esse tipo de procedimento seria tratado perfeitamente pelo antigo programa de gravação de macros do Lotus 1-2-3, lançado em 1983. Mitch Kapor resolveu esse problema há 21 anos e a Microsoft ainda não conseguiu corrigi-lo.

Você sabia que, até o Excel 97, a Microsoft Excel executava secretamente as macros de linha de comando do Lotus? Eu não sabia, mas descobri logo depois que a Microsoft parou de dar suporte ao Excel 97. Na época, diversas empresas atualizavam para o Excel XP, que não mais suportava as macros do Lotus 1-2-3. Muitas dessas empresas nos contratavam para converter as antigas macros do Lotus 1-2-3 em macros do Excel VBA. É interessante que, a partir do Excel 5, Excel 95 e Excel 97, a Microsoft começou a oferecer um interpretador que podia tratar as macros do Lotus que resolviam esse problema corretamente, embora o próprio programa de gravação de macros da Microsoft não consiga (mesmo hoje em dia!) resolver o problema.

Próximos passos: aprender VBA é a solução

No Capítulo 2, "Isso parece BASIC, então por que não parece familiar?", examinamos as duas macros que gravamos em um esforço de fazer com que tivessem algum sentido. Depois de entender como decodificar o código do VBA, será natural corrigir o código gravado ou simplesmente escrever código a partir do zero. Espere o próximo capítulo e você poderá escrever um código útil que funciona consistentemente.

Isso parece o BASIC, então por que não parece familiar?

2

Não consigo entender esse código

Como mencionei anteriormente, se você já estudou uma linguagem procedural como BASIC ou COBOL, pode ser realmente confuso quando examinar o código VBA. É verdade, VBA significa Visual Basic for Applications, mas é uma versão *orientada a objetos* do BASIC. Eis um fragmento do código VBA:

```
Selection.End(xlDown).Select
Range("A14").Select
ActiveCell.FormulaR1C1 = "'Total"
Range("E14").Select
Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
Selection.AutoFill _
   Destination:=Range("E14:G14"), _
   Type:=xlFillDefault
```

Esse código provavelmente não fará sentido para alguém que conhece apenas linguagens procedurais e, infelizmente, suas primeiras noções de programação (supondo que você tenha mais de 25 anos) devem ter sido em uma linguagem procedural.

Eis uma seção do código escrito na linguagem BASIC:

```
For x = 1 to 10
   Print Rpt$(" ",x);
   Print "*"
Next x
```

Se executar esse código, você terá uma pirâmide de asteriscos na tela:

```
*
 *
  *
   *
    *
     *
      *
       *
        *
         *
```

Se já fez um curso de programação procedural, é provável que você possa examinar o código e descobrir o que está acontecendo. Acho que uma linguagem procedural é mais parecida com o inglês do que com linguagens orientadas a objetos. A instrução `Print "Ola Mundo!"` segue o formato de verbo-objeto e, em geral, é assim que se fala na língua inglesa. Vamos parar de discutir programação por um segundo e pensar em um exemplo concreto.

Entendendo as partes da "sintaxe" do VBA

Se você fosse jogar futebol utilizando o BASIC, a instrução chutar a bola seria algo como

"Chute a bola"

NESTE CAPÍTULO

Ei — é assim que falamos! Isso faz sentido. Há um verbo (chutar) e depois um substantivo (a bola). No código BASIC da seção anterior, você tem um verbo (imprimir) e um substantivo (um asterisco). A vida é boa.

Eis o problema. O VBA não funciona assim. Nenhuma linguagem orientada a objetos funciona dessa maneira. Em uma linguagem orientada a objetos, os objetos (os substantivos) são mais importantes (daí, o nome: orientado a objeto). Se você for jogar futebol com o VBA, a estrutura básica seria:

Bola.Chuta

Há um substantivo — a bola. Ele vem primeiro. No VBA, isso é um *objeto*. Então, você tem o verbo — chutar. Ele vem depois. No VBA, isso é um *método*.

A estrutura básica do VBA é um grupo de linhas de código em que você tem

Objeto.Método

Sinto muito, isso não é inglês. Se estudou línguas latinas na escola, você se lembrará de que elas utilizam uma construção do tipo 'substantivo adjetivo', mas não conheço ninguém que fale 'substantivo verbo' ao pedir para alguém fazer algo. Você fala dessa maneira?

```
Water.Drink
Food.Eat
Girl.Kiss
```

É claro que não. Essa é a razão pela qual o VBA é tão confuso para alguém que já fez um curso de programação procedural.

Vamos continuar um pouco mais com a analogia. Imagine que você esteja caminhando em um campo gramado e que haja cinco bolas à sua frente. Há uma bola de futebol, uma de basquete, uma de beisebol, uma de boliche e uma de tênis. Você quer instruir um menino do time de futebol a

Chutar a bola de futebol

Se pedir para ele chutar a bola (ou `ball.kick`), você não estará tão certo de qual ele chutará. Talvez ele chute a bola mais próxima dele. Isso poderia ser um problema real se ele estivesse na frente da bola de boliche.

Para quase todos os substantivos, ou objetos, no VBA, há uma coleção desses objetos. Pense no Excel. Se houver uma linha, você poderá ter algumas linhas. Se houver uma célula, você poderá ter algumas células. Se houver uma planilha, você poderá ter algumas planilhas. A única diferença entre um objeto e uma coleção é que você adicionará um *s* ao nome do objeto:

A linha torna-se linhas

A célula torna-se células

A bola torna-se bolas

Ao se referir a algo que é uma coleção, você precisa dizer à linguagem de programação a cujo item você está se referindo. Há duas maneiras de fazer isso. Você pode referenciar um item utilizando um número:

```
Balls(2).Kick
```

O que funcionará bem, mas parece uma maneira perigosa de programar. Isso pode funcionar terça-feira; mas se quarta-feira você voltar ao campo e alguém tiver reorganizado as bolas, Bolas(2).Chutar pode ser um exercício difícil.

A outra maneira é utilizar um nome para o objeto. Para mim, essa é uma opção bem mais segura. Você pode dizer:

```
Balls("Soccer").Kick
```

Com esse método, você sempre saberá que uma bola de futebol é que será chutada.

Até aqui, tudo bem. Você sabe que poderá chutar uma bola e que ela será a de futebol. Para a maioria dos verbos, ou métodos, no Excel VBA, há parâmetros que dizem *como* fazer a ação. Esses atuam como advérbios. Talvez você queira que a bola de futebol seja chutada para a esquerda e com muita força. A maioria dos métodos tem diversos parâmetros que dizem como o programa deve realizar o método:

```
Balls("Soccer").Kick Direction:=Left, Force:=Hard
```

Ao examinar o código VBA, quando vir a combinação dois-pontos/sinal de igual, você sabe que está examinando parâmetros sobre como o verbo deve ser executado.

Às vezes, um método terá uma lista de dez parâmetros. Alguns podem ser opcionais. Talvez o método `Kick` tenha um parâmetro `Elevation`. Talvez você tenha esta linha de código:

```
Balls("Soccer").Kick Direction:=Left, Force:=Hard, Elevation:=High
```

Eis a parte radicalmente confusa. Cada método tem uma ordem padrão para seus parâmetros. Se não for um programador cuidadoso, e caso conheça a ordem dos parâmetros, você poderá omitir os nomes de parâmetro. O seguinte código é equivalente à linha anterior do código.

```
Balls("Soccer").Kick Left, Hard, High
```

Isso dificulta nosso entendimento. Sem os dois-pontos/sinal de igual, não é óbvio que temos parâmetros. A menos que conheça a ordem do parâmetros, talvez você não entenda o que está sendo dito. É muito fácil com Left (para a esquerda), Hard (com força) e High (para o alto), mas, quando há parâmetros como o seguinte:

```
WordArt.Add Left:=10, Top:=20, Width:=100, Height:=200
```

fica realmente confuso visualizar:

```
WordArt.Add 10, 20, 100, 200
```

O anterior é um código válido, mas a menos que você saiba que a ordem padrão dos parâmetros para esse método Adicionar é Esquerda, Parte Superior, Largura, Altura, esse código não fará sentido. A ordem padrão para cada método em particular é a ordem dos parâmetros, como mostrado no tópico da ajuda para esse método.

Para tornar a vida mais confusa, você pode começar especificando parâmetros na ordem padrão sem nomeá-los e, então, de repente, mudar para a nomeação de parâmetros quando surgir um que não corresponda à ordem padrão. Se você quiser chutar a bola para a esquerda e para cima sem se preocupar com a força (você está disposto a aceitar a força padrão), as duas instruções a seguir são equivalentes:

```
Balls("Soccer").Kick Direction:=Left, Elevation:=High
Balls("Soccer").Kick Left, Elevation:=High
```

Assim que você começar a nomear os parâmetros, eles têm de ser nomeados em toda essa linha de código.

Alguns métodos simplesmente funcionam por conta própria. Para simular o pressionamento da tecla F9, utilize este código:

```
Application.Calculate
```

Outros métodos realizam uma ação e criam algo. Por exemplo, você pode adicionar uma planilha com

```
Worksheets.Add Before:=Worksheets(1)
```

Entretanto, como Worksheets.Add na verdade cria um novo objeto, você pode atribuir os resultados desse método a uma variável. Nesse caso, você deve colocar os parâmetros entre parênteses:

```
Set MyWorksheet = Worksheets.Add(Before:=Worksheets(1))
```

Só mais um pouco de gramática é necessário. Adjetivos. Os adjetivos descrevem um substantivo. As propriedades descrevem um objeto. Aqui, todos somos fãs do Excel; então vamos trocar a analogia do futebol por uma analogia com o Excel. Há um objeto para descrever a célula ativa. Felizmente, esse objeto tem o nome bem descritivo de

```
ActiveCell
```

Vamos supor que quiséssemos mudar a cor da célula ativa para amarelo. Há uma propriedade chamada `InteriorColor` para uma célula. Ela usa uma série complexa de códigos, mas é possível mudar a cor da célula para amarelo utilizando este código:

```
ActiveCell.Interior.ColorIndex = 6
```

Você está confuso? Percebe como isso é tão complicado? Mais uma vez, há a construção Substantivo-Ponto-Algo, mas dessa vez é `Object.Property` e não `Object.Method`.

Como diferenciá-los? Bem, é bastante sutil. Não há nenhum sinal de dois-pontos antes do sinal de igual. Uma propriedade é quase sempre definida como igual a alguma coisa ou talvez o valor de uma propriedade esteja sendo atribuído a outra coisa.

Para que a cor dessa célula seja a mesma da célula A1, talvez você diga que

```
ActiveCell.Interior.ColorIndex = Range("A1").Interior.ColorIndex
```

`Interior.ColorIndex` é uma propriedade. Mudando o valor de uma propriedade, você pode mudar a aparência das coisas. É esquisito — mude um adjetivo e você realmente altera algo na célula. Humanos diriam: "pinte a célula de amarelo". O VBA diria

```
ActiveCell.Interior.ColorIndex = 30
```

A Tabela 2.1 resume as 'partes da fala' do VBA.

Tabela 2.1 Partes da linguagem de programação do VBA

Componente VBA	Análogo a	Observações
Objeto	Substantivo	
Coleção	Substantivo plural	Normalmente especifica qual objeto: Worksheets(1).
Método	Verbo	`Objeto.Método.`
Parâmetro	Advérbio	Lista os parâmetros depois do método. Separe o nome de parâmetro do valor com :=.
Propriedade	Adjetivo	Você pode configurar uma propriedade `activecell.height = 10` ou consultar o valor de uma propriedade `x = activecell.height`.

O VBA é tão difícil assim? Não!

Saber se você está lidando com propriedades ou métodos ajudará a definir a sintaxe certa para seu código. Não se preocupe se tudo isso agora parece confuso. Ao escrever o código VBA a partir do zero, é difícil saber se o processo de mudar a cor de uma célula para amarelo exige um verbo ou apenas um adjetivo. É um método ou uma propriedade?

É *nisso* que está a beleza do programa de gravação de macros. Quando não sabemos como codificar algo, gravamos uma macro bem pequena, examinamos o código gravado e descobrimos o que está acontecendo.

Arquivos de ajuda do VBA — usando F1 para localizar qualquer coisa

Esse é um recurso muito legal, mas primeiro vamos examinar uma situação mais complexa Se for escrever macros VBA, você realmente *precisará* instalar os tópicos de ajuda do VBA. O problema: os tópicos de ajuda do VBA não são instalados na instalação do Office padrão. Eis como ver se você tem ou não esses tópicos.

1. Abra o Excel e acesse o Editor do VB pressionando Alt+F11. No menu Inserir, selecione Módulo (veja Figura 2.1).

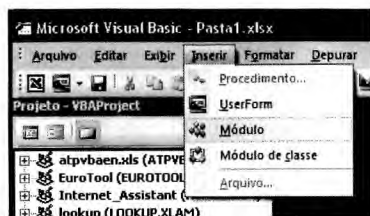
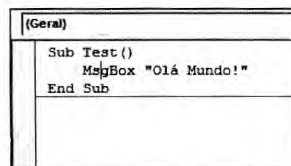
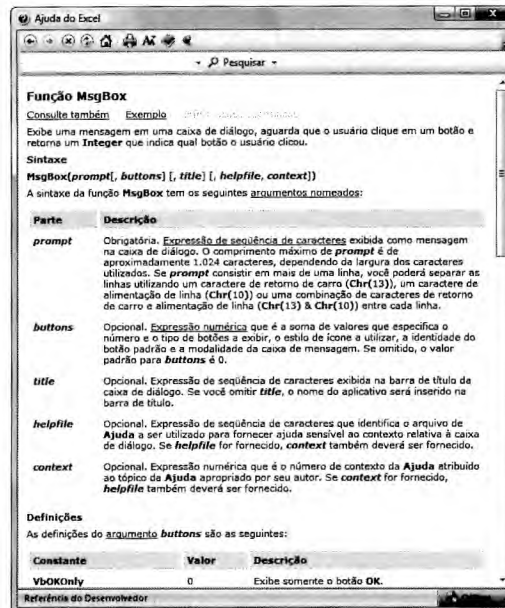
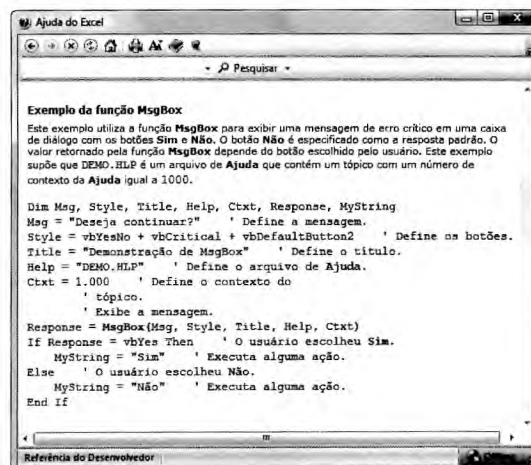
Figura 2.1
Insira um novo módulo na pasta de trabalho em branco.

2. Digite as três linhas de código mostradas na Figura 2.2. Clique dentro da palavra `MsgBox`.

Figura 2.2
Clique dentro da palavra *MsgBox* e pressione F1.

3. Com o cursor na palavra `MsgBox`, pressione a tecla F1. Se os tópicos de ajuda do VBA estiverem instalados, você verá o tópico de ajuda mostrado na Figura 2.3.

Se uma mensagem informar que não há ajuda sobre esse tópico, você precisará encontrar os CDs originais (ou obter os direitos para a pasta de instalação com o administrador de rede) para poder instalar os tópicos de ajuda do VBA. Examine o processo de fazer uma reinstalação. Durante a reinstalação, selecione a instalação personalizada e certifique-se de selecionar os arquivos de ajuda do VBA.

Figura 2.3
Se os tópicos de ajuda do VBA tiverem sido instalados, você verá esta tela.

Utilizando os tópicos de ajuda

Se solicitar ajuda sobre uma função ou um método, a ajuda mostrará os vários argumentos disponíveis. Se navegar até a parte inferior da maioria dos tópicos de ajuda, a ajuda apresentará exemplos de código sob o título Exemplo. Essa seção de cada tópico de ajuda é um excelente recurso (veja Figura 2.4).

Figura 2.4
A maioria dos tópicos de ajuda apresenta um exemplo.

É possível selecionar o código e copiá-lo para a área de transferência pressionando Ctrl+C (veja Figura 2.5), e depois colá-lo no módulo pressionando Ctrl+V.

Depois de gravar uma macro, sem dúvida haverá objetos ou métodos dos quais você não está seguro. Insira o cursor do mouse em qualquer palavra-chave e pressione F1 para obter ajuda sobre esse tópico.

Figura 2.5
Destaque o código no arquivo de ajuda e copie-o com Ctrl+C.

Examinando o código de macro gravada — utilizando o Editor e a Ajuda do VB

Vamos examinar o código que foi gravado quando trabalhávamos no exemplo do Capítulo 1 para ver se ele agora faz mais sentido no contexto de objetos, propriedades e métodos. Você também pode ver se é ou não possível corrigir os erros produzidos pelo programa de gravação de macros.

Eis o primeiro código que o Excel gravou no exemplo no Capítulo 1 (veja Figura 2.6).

Figura 2.6
Código gravado no exemplo do Capítulo 1.

```
(Geral)                                  ImportInvoice

Sub ImportInvoice()
'
' Macro ImportarFatura
' Macro gravada em 23/10/2003 por Bill Jelen Essa macro importará fatura.txt and adicionará os totais.
'
' Atalho de teclado: Ctrl+i
'
    Workbooks.OpenText Filename:= _
        "C:\fatura.txt", Origin _
        :=437, StartRow:=1, DataType:=xlDelimited, TextQualifier:=xlDoubleQuote _
        , ConsecutiveDelimiter:=False, Tab:=True, Semicolon:=False, Comma:=True _
        , Space:=False, Other:=False, FieldInfo:=Array(Array(1, 3), Array(2, 1), _
        Array(3, 1), Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)), TrailingMinusNumbers _
        :=True
    Selection.End(xlDown).Select
    Range("A14").Select
    ActiveCell.FormulaR1C1 = "'Total"
    Range("E14").Select
    Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
    Selection.AutoFill Destination:=Range("E14:G14"), Type:=xlFillDefault
    Range("E14:G14").Select
    Rows("1:1").Font.Bold = True
    Rows("14:14").Select
    Selection.Font.Bold = True
    Cells.Select
    Selection.Columns.AutoFit
End Sub
```

Agora que você entende o conceito de Substantivo.Verbo ou Objeto.Método, analise a primeira linha do código. Digamos, `Workbooks.OpenText`. Nesse caso, `Workbooks` é um objeto. `OpenText` é um método. Clique com o cursor do mouse na palavra *`OpenText`* e pressione F1 para obter uma explicação do método `OpenText` (veja Figura 2.7).

Ajuda do Excel

Referência do Desenvolvedor do Excel 2007 > Referência da modelos de objeto do Excel > Objeto Workbooks > Métodos

Referência sobre o desenvolvedor do Excel

Workbooks.Método OpenText

Carrega e analisa um arquivo de texto como uma nova pasta de trabalho com uma única planilha contendo os dados do arquivo de texto analisados.

Sintaxe

expressão.**OpenText**(***Filename, Origin, StartRow, DataType, TextQualifier, ConsecutiveDelimiter, Tab, Semicolon, Comma, Space, Other, OtherChar, FieldInfo, TextVisualLayout, DecimalSeparator, ThousandsSeparator, TrailingMinusNumbers, Local***)

expressão Uma variável que representa um objeto **Workbooks**.

Parâmetros

Nome	Obrigatório/Opcional	Tipo de dados	Descrição
Filename	Obrigatório	**Seqüência**	Especifica o nome do arquivo de texto a ser aberto e analisado.
Origin	Opcional	**Variant**	Especifica a origem do arquivo de texto. Pode ser uma das seguintes constantes **XlPlatform: xlMacintosh, xlWindows** ou **xlMSDOS**. Além disso, também pode ser um inteiro representando o número da página de código desejada. Por exemplo, "1256" especifica que a codificação do arquivo de origem seja Arabic (Windows). Se esse argumento for omitido, o método usará a configuração atual da opção **Origem do Arquivo** da caixa de diálogo **Assistente de Importação de Texto**.
StartRow	Opcional	**Variant**	O número da linha por onde começar a análise do texto. O valor padrão é 1.
DataType	Opcional	**Variant**	Especifica o formato de colunas dos dados do arquivo. pode ser uma das seguintes constantesXlTextParsingType: **xlDelimited** ou **xlFixedWidth**. Se este argumento não for especificado, o Microsoft Excel tentará determinar o formato de colunas quando abrir o arquivo.

Referência do Desenvolvedor   Offline

Figura 2.7
Tópico de ajuda para o método `OpenText`.

O arquivo de ajuda confirma se `OpenText` é um método ou uma palavra de ação. Na caixa cinza está a ordem padrão para todos os argumentos que podem ser utilizados com `OpenText`. Note que apenas um argumento é necessário: `FileName`. Todos os outros argumentos são listados como opcionais.

Parâmetros opcionais

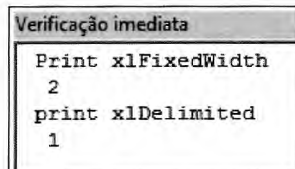
O que acontece se você pular um parâmetro opcional? O arquivo de ajuda pode informar isso. Para `StartRow`, o arquivo de ajuda indica que o valor padrão é 1. Se você omitir o parâmetro `StartRow`, o Excel começa importando na linha 1. Isso é relativamente seguro. Examine a nota do arquivo de ajuda sobre `Origin`. Se esse argumento for omitido, você herdará o valor utilizado para `Origin` da última vez que alguém utilizou esse recurso no Excel nesse computador. Sem dúvida, essa é uma receita para desastres — o código pode funcionar 98 por cento das vezes, mas, assim que alguém importar um arquivo em árabe, o Excel lembrará da configuração árabe e irá supor que é isso que a macro quer caso você não codifique esse parâmetro explicitamente.

Constantes definidas

Analise a entrada do arquivo de ajuda para `DataType` na Figura 2.7. O arquivo de ajuda diz que pode ser uma destas constantes: `xlDelimited` ou `xlFixedWidth`. O arquivo de ajuda informa que essas são as constantes `xlTextParsingType` válidas. Essas constantes são predefinidas no Excel VBA. No Editor do VB, pressione Ctrl+G para abrir a janela Verificação Imediata. Na janela Verificação Imediata, digite esta linha e pressione Enter:

```
Print xlFixedWidth
```

A resposta aparece na janela Verificação Imediata. `xlFixedWidth` é o equivalente a dizer '2' (veja Figura 2.8). Solicite que a janela Verificação Imediata execute o comando `Print xlDelimited`. Na realidade isso é o mesmo que digitar 1. A Microsoft supõe, corretamente, que é mais fácil alguém ler um código que utiliza o termo `xlDelimited` semelhante ao inglês do que ler 1.

Figura 2.8
Na janela Verificação Imediata do Editor do VB, pesquise para ver o valor real de constantes como `xlFixedWidth`.

Se fosse um programador maldoso, com certeza você poderia memorizar todas essas constantes e escreveria o código utilizando os equivalentes numéricos das constantes. Mas os deuses da programação (bem como a pessoa ao seu lado que precisa examinar seu código) o amaldiçoarão por isso.

Na maioria dos casos, o arquivo de ajuda exige especificamente os valores válidos das constantes ou oferece um hiperlink azul que expande o arquivo de ajuda e mostra os valores válidos das constantes (veja Figura 2.9).

Enumeração XlColumnDataType

Especifica Especifica como uma coluna deve ser analisada.

Informação sobre versão
Version adicionada: Excel 2007

Nome	Valor	Descrição
xlDMYFormat	4	Formato de data DMY.
xlDYMFormat	7	Formato de data DYM.
xlEMDFormat	10	Formato de data EMD.
xlGeneralFormat	1	Geral.
xlMDYFormat	3	Formato de data MDY.
xlMYDFormat	6	Formato de data MYD.
xlSkipColumn	9	A coluna não é analisada.
xlTextFormat	2	Texto.
xlYDMFormat	8	Formato de data YDM.
xlYMDFormat	5	Formato de data YMD.

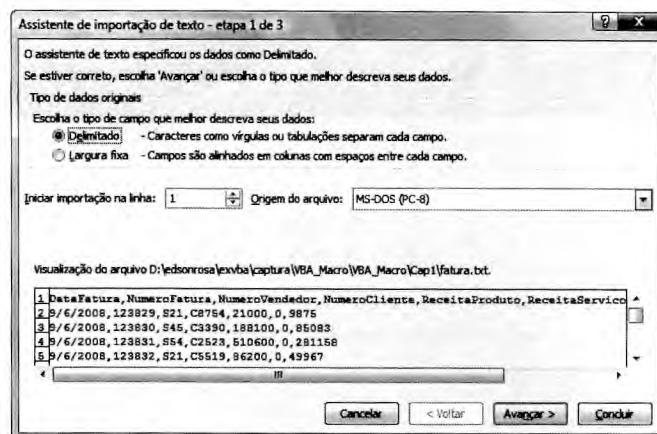
© 2006 Microsoft Corporation. Todos os direitos reservados.

Figura 2.9
Clique no hiperlink azul para ver todos os possíveis valores de constantes. Aqui, as dez possíveis constantes `xlColumnDataType` são exibidas em um novo tópico de ajuda.

Minha única reclamação em relação a esse excelente sistema de ajuda é que ele não identifica os parâmetros que podem ser novos em uma dada versão. Nesse caso particular, `TrailingMinusNumbers` foi introduzido no Excel 2002. Se tentar dar esse programa a alguém que ainda utiliza o Excel 2000, o código não executará porque ele não entende o parâmetro `TrailingMinusNumbers`. Esse é um problema frustrante e a única maneira de aprender a tratá-lo é por meio de tentativa e erro.

Se você ler o tópico de ajuda em `OpenText`, pode supor que ele, basicamente, é o equivalente a abrir um arquivo utilizando o Assistente de Importação de Texto. No primeiro passo do assistente, em geral você escolhe Delimitado ou Largura Fixa. Você também especifica a Origem do Arquivo e em que linha deve iniciar (veja Figura 2.10). Essa primeira etapa do assistente é tratada por esses parâmetros do método `OpenText`:

```
Origin:=437
StartRow:=1
DataType:=xlDelimited
```

Figura 2.10
O primeiro passo do Assistente de Importação de Texto no Excel abrange três parâmetros do método `OpenText`.

O segundo passo do assistente Texto para Colunas permite especificar que os campos devem ser separados por uma vírgula. Não é recomendável tratar duas vírgulas como uma única vírgula, assim a opção Considerar Delimitadores Consecutivos Como Um Só está desmarcada. Às vezes, um campo pode conter uma vírgula — por exemplo "XYZ, Inc.". Nesse caso, o campo deve ter aspas em torno do valor. Isso é especificado na caixa Qualificador de Texto (veja Figura 2.11). Esse passo do assistente *é tratado por estes* parâmetros do método `OpenText`:

```
TextQualifier:=xlDoubleQuote
ConsecutiveDelimiter:=False
Tab:=False
Semicolon:=False
Comma:=True
Space:=False
Other:=False
```

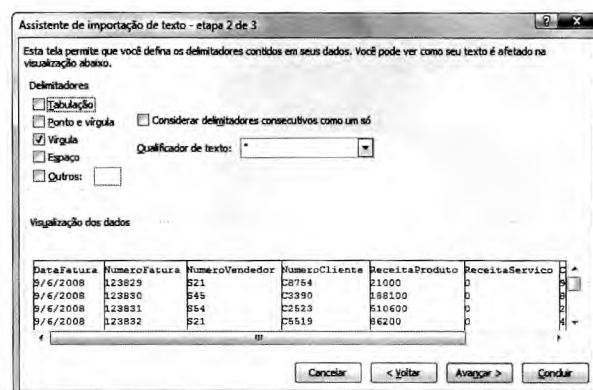
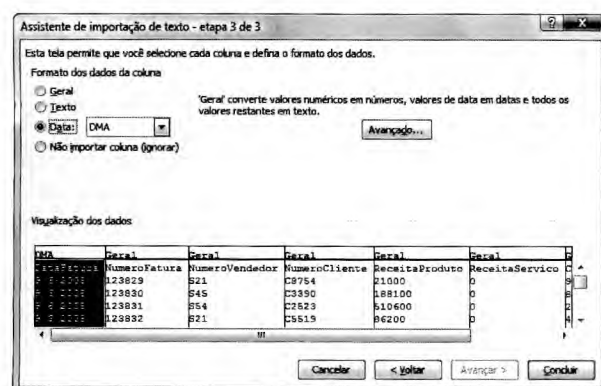
Figura 2.11
O segundo passo do Assistente de Importação de Texto é tratado pelos sete parâmetros do método `OpenText`.

O terceiro passo do assistente é onde você realmente identifica os tipos de campo. No nosso caso, deixamos todos os campos como Geral, exceto o primeiro, que foi marcado como uma data no formato MDA (Mês, Dia, Ano) (veja Figura 2.12). Isso é representado no código pelo parâmetro `FieldInfo`.

Figura 2.12
O terceiro passo do Assistente de Importação de Texto é relativamente complexo. O parâmetro `FieldInfo` inteiro do método `OpenText` duplica as escolhas feitas nesse passo do assistente.

Se por acaso clicar no botão Avançado no terceiro passo do assistente, você terá a oportunidade de especificar algo além do Separador Decimal e de Milhar padrão, e também a configuração para Sinal de Subtração à direita para Números Negativos (veja *Figura* 2.13). Observe que o programa de gravação de macros não escreve código para `DecimalSeparator` ou `ThousandsSeparator`, a menos que você os altere a partir dos padrões. O programa de gravação de macros sempre grava o parâmetro `TrailingMinusNumbers`.

Nesse ponto, é possível ver mais ou menos como quase todas as opções que você faz no Excel durante a execução do programa de gravação de macros correspondem a um fragmento de código na macro gravada.

Eis outro exemplo. A próxima linha de código na macro é

```
Selection.End(xlDown).Select
```

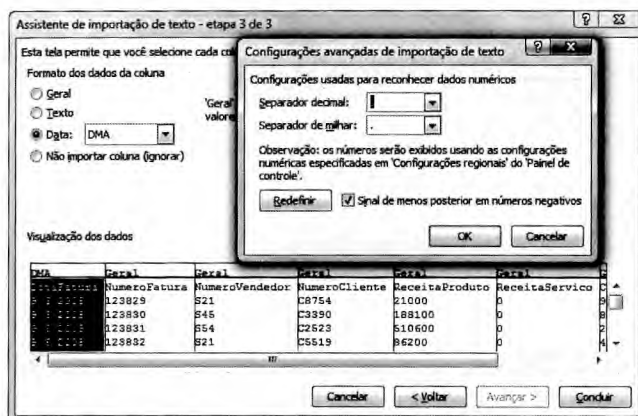
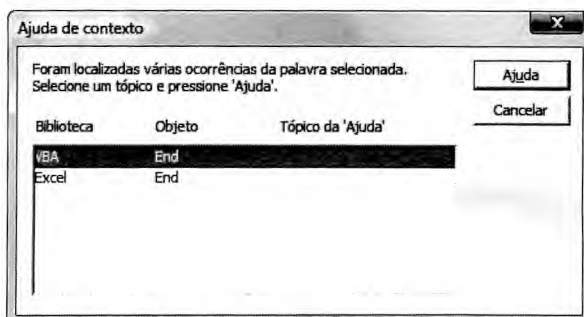
Figura 2.13
O parâmetro `TrailingMinusNumbers` vem da opção Configurações Avançadas de Importação de Texto. Se tivesse alterado um dos campos do separador, os novos parâmetros seriam gravados pelo programa de gravação de macros.

Você pode clicar e obter ajuda para três tópicos nesta linha de código: `Selection`, `End` e `Select`. Supondo que `Selection` e `Select` sejam nomes relativamente auto-explicativos, clique na palavra *End* e pressione F1 para ajuda. Uma caixa de diálogo Ajuda de Contexto abre e informa que há dois possíveis tópicos de ajuda para `End`. Há um na biblioteca do Excel e um na biblioteca do VBA (veja Figura 2.14).

Figura 2.14
Às vezes ao clicar em uma palavra-chave do VBA e pressionar F1, você precisa 'adivinhar' qual biblioteca de ajuda utilizar.

Se for iniciante no VBA, poderá ser frustrante decidir qual selecionar. Faça a melhor suposição e clique em Ajuda. Nesse caso, o tópico de ajuda para End na biblioteca do VBA está relacionado à instrução `End` (veja Figura 2.15), o que não é o que temos aqui.

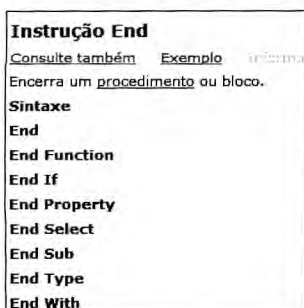
Instrução End

Consulte também Exemplo

Encerra um procedimento ou bloco.

Sintaxe

End
End Function
End If
End Property
End Select
End Sub
End Type
End With

Figura 2.15
Se tiver feito a escolha errada, o tópico de ajuda exibido será outro completamente diferente. É fácil tentar novamente.

Feche a Ajuda, pressione F1 novamente e escolha o objeto `End` na biblioteca do Excel. Esse tópico de ajuda diz que `End` é uma propriedade. Ele retorna um objeto `Range`, que é equivalente a pressionar End+Up ou End+Down na interface do Excel (veja Figura 2.16). Se clicar no hiperlink azul para `xlDirection`, você verá os parâmetros válidos que podem ser passados para a função `End`.

Referência sobre o desenvolvedor do Excel

Range.Propriedade End

Retorna um objeto **Range** representando a célula no fim da região que contém o intervalo de origem. Equivalente a pressionar END+SETA PARA CIMA, END+SETA PARA BAIXO, END+SETA PARA A ESQUERDA ou END+SETA PARA A DIREITA. Objeto **Range** de somente leitura.

Sintaxe

expressão.**End(*Direction*)**

expressão Uma variável que representa um objeto **Range**.

Parâmetros

Nome	Obrigatório/Opcional	Tipo de dados	Descrição
Direction	Obrigatório	**XlDirection**	A direção na qual mover.

Exemplo

Este exemplo seleciona a célula no topo da coluna B na região que contém a célula B4.

Visual Basic for Applications

```
Range("B4").End(xlUp).Select
```

Este exemplo seleciona a célula no final da linha 4 na região que contém a célula B4.

Visual Basic for Applications

```
Range("B4").End(xlToRight).Select
```

Figura 2.16
O tópico de ajuda correto para a propriedade `End`.

Propriedades podem retornar objetos

Na discussão inicial neste capítulo, afirmamos que a sintaxe básica do VBA era `Object.Method`. Nessa linha específica do código, o método é `Select`. A palavra-chave `End` é uma propriedade, mas no arquivo de ajuda você vê que ela retorna um objeto Range. Como o método `Select` pode ser aplicado a um objeto Range, na verdade o método é acrescentado a uma propriedade.

Você poderia então assumir que `Selection` é o objeto nessa linha de código. Se clicar na palavra `Selection` e pressionar F1, você verá que, de acordo com o tópico da ajuda, `Selection` é uma propriedade e não um objeto. Na realidade, o código adequado seria dizer `Application.Selection`; entretanto, ao executá-lo no Excel, o VBA supõe que você está se referindo ao modelo de objeto do Excel, então você pode omitir o objeto Application. Se você fosse escrever um programa no Word VBA para automatizar o Excel, precisaria incluir uma variável de objeto antes da propriedade `Selection` para qualificar o aplicativo ao qual estivesse se referindo.

Nesse caso, o `Application.Selection` pode retornar diversos tipos de objetos diferentes. Se uma célula for selecionada, ela retornará o objeto Range.

Utilizando ferramentas de depuração para entender o código gravado

O Editor do Visual Basic apresenta algumas ferramentas de depuração impressionantes. Essas ferramentas são excelentes para ajudá-lo a ver o que o código da macro gravada está fazendo.

Percorrendo o código

Em geral, uma macro executa com uma grande rapidez. Você a inicia e, em menos de um segundo, ela foi executada. Se algo der errado, não há como descobrir o que ela está fazendo. Utilizando o recurso Depuração Total do Excel, é possível executar uma linha de código por vez.

Certifique-se de que o cursor está no procedimento `ImportInvoice` e, no menu, selecione Depurar, Depuração Total, como mostrado na Figura 2.17 (ou pressione F8).

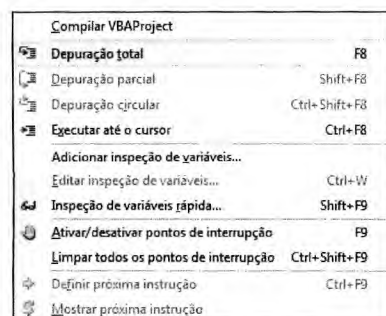
Figura 2.17
A Depuração Total permite executar uma linha por vez.

O Editor do VB agora está no modo Depuração. A linha prestes a ser executada é destacada em amarelo com uma seta na margem antes do código (veja Figura 2.18).

Figura 2.18
A primeira linha da macro está prestes a ser executada.

```
Sub ImportInvoice()
'
' ImportInvoice Macro
'
' Atalho pelo teclado: Ctrl+i
'
'    Workbooks.OpenText Filename:= _
'        "fatura.txt", Origin _
'        :=437, StartRow:=1, DataType:=xlDelimit
'        , ConsecutiveDelimiter:=False, Tab:=Tru
'        , Space:=False, Other:=False, FieldInfo
'        Array(3, 1), Array(4, 1), Array(5, 1),
'        :=True
    Selection.End(xlDown).Select
    Range("A14").Select
    ActiveCell.FormulaR1C1 = "'Total"
    Range("E14").Select
    Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]
    Selection.AutoFill Destination:=Range("E14:
    Range("E14:G14").Select
    Rows("1:1").Font.Bold = True
    Rows("14:14").Select
    Selection.Font.Bold = True
    Cells.Select
    Selection.Columns.AutoFit
End Sub
```

Nesse caso, a próxima linha a ser executada é a linha `Sub ImportInvoice()`. Basicamente, isso informa: 'estamos prestes a começar a executar esse procedimento'. Pressione a tecla F8 para executar a linha em amarelo e passe para a próxima linha do código. O longo código para `OpenText` é então destacado. Pressione F8 para executar essa linha do código. Quando vir que `Selection.End(xlDown).Select` foi destacado, você saberá que o Visual Basic terminou de executar o comando `OpenText`. De fato, você pode pressionar Alt+Tab para alternar para o Excel e ver se o arquivo `Fatura.txt` foi analisado sintaticamente no Excel. Observe que A1 está selecionada (veja Figura 2.19).

Figura 2.19
Alternar para o Excel mostra que o arquivo `fatura.txt` foi de fato importado.

	A	B	C	D	E	F	G	H
1	DataFatura	NumeroFatu	NumeroVende	NumeroClien	ReceitaProd	ReceitaSer	CustoProduto	
2	06/07/2008	123829	S21	C8754	538400	0	299897	
3	06/07/2008	123830	S45	C4056	588600	0	307563	
4	06/07/2008	123831	S54	C8323	882200	0	521726	
5	06/07/2008	123832	S21	C6026	830900	0	494831	
6	06/07/2008	123833	S45	C3025	673600	0	374953	
7	06/07/2008	123834	S54	C8663	966300	0	528575	
8	06/07/2008	123835	S21	C1508	467100	0	257942	
9	06/07/2008	123836	S45	C7366	658500	10000	308719	
10	06/07/2008	123837	S54	C4533	191700	0	109534	
11								

Volte ao Editor do VB pressionando Alt+Tab. A próxima linha prestes a ser executada é `Selection.End(xlDown).Select`. Pressione F8 para executar esse código. Alterne para o Excel a fim de ver os resultados. Agora A10 está selecionada (veja Figura 2.20).

Figura 2.20
Verifique se o comando `End(xlDown).Select` funcionou da maneira esperada. Isso é o equivalente a pressionar a tecla End e, então, a seta para baixo.

	A	B	C	D	E	F	G	H
1	DataFatura	NumeroFatu	NumeroVende	NumeroClien	ReceitaProd	ReceitaSer	CustoProduto	
2	06/07/2008	123829	S21	C8754	538400	0	299897	
3	06/07/2008	123830	S45	C4056	588600	0	307563	
4	06/07/2008	123831	S54	C8323	882200	0	521726	
5	06/07/2008	123832	S21	C6026	830900	0	494831	
6	06/07/2008	123833	S45	C3025	673600	0	374953	
7	06/07/2008	123834	S54	C8663	966300	0	528575	
8	06/07/2008	123835	S21	C1508	467100	0	257942	
9	06/07/2008	123836	S45	C7366	658500	10000	308719	
10	06/07/2008	123837	S54	C4533	191700	0	109534	
11								

Pressione F8 novamente para executar a linha `Range("A14").Select`. Se alternar para o Excel pressionando Alt+Tab, você verá que é nesse ponto que a macro começa a apresentar problemas. Em vez de passar para a primeira linha em branco, o programa passou para a linha errada (veja Figura 2.21).

Figura 2.21
O código da macro gravada move-se cegamente até a Linha 14 da linha Total.

	A	B	C	D	E	F	G	H
1	DataFatura	NumeroFatu	NumeroVende	NumeroClien	ReceitaProd	ReceitaSer	CustoProduto	
2	06/07/2008	123829	S21	C8754	538400	0	299897	
3	06/07/2008	123830	S45	C4056	588600	0	307563	
4	06/07/2008	123831	S54	C8323	882200	0	521726	
5	06/07/2008	123832	S21	C6026	830900	0	494831	
6	06/07/2008	123833	S45	C3025	673600	0	374953	
7	06/07/2008	123834	S54	C8663	966300	0	528575	
8	06/07/2008	123835	S21	C1508	467100	0	257942	
9	06/07/2008	123836	S45	C7366	658500	10000	308719	
10	06/07/2008	123837	S54	C4533	191700	0	109534	
11								
12								
13								
14								
15								

Agora que a área do problema foi identificada, você pode parar a execução do código utilizando o *comando* Redefinir (selecione Executar, Redefinir ou clique no botão Redefinir na barra de ferramentas; veja Figura 2.22). Depois de clicar em Redefinir, você deve retornar ao Excel e desfazer tudo que foi feito pela macro parcialmente completada. Nesse caso, feche o arquivo `fatura.txt` sem salvar.

Figura 2.22
O botão Redefinir na barra de ferramentas pára uma macro que está no modo Interrupção.

Mais opções de depuração — pontos de interrupção

Se tiver centenas de linhas de código, você pode não querer analisar todas individualmente. Talvez saiba que o problema está em determinada seção do programa. Nesse caso, pode definir um ponto de interrupção. Você pode então iniciar a execução do código, mas a macro interrompe um pouco antes de ela executar a linha do ponto de interrupção do código.

Para configurar um ponto de interrupção, clique na área cinza da margem à esquerda da linha do código que você quer parar. Um grande ponto marrom aparece ao lado desse código e a linha do código é destacada em marrom (veja Figura 2.23).

Figura 2.23
O grande ponto marrom denota um ponto de interrupção.

```
Sub ImportInvoice()
'
' ImportInvoice Macro
'
' Atalho de teclado: Ctrl+i
'
'    Workbooks.OpenText Filename:= _
'        "fatura.txt", Origin _
'        :=437, StartRow:=1, DataType:=xlDelimited, TextQualifier:=xlDoubleQuote _
'        , ConsecutiveDelimiter:=False, Tab:=True, Semicolon:=False, Comma:=True _
'        , Space:=False, Other:=False, FieldInfo:=Array(Array(1, 3), Array(2, 1), _
'        Array(3, 1), Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)), TrailingMinusNumbers _
'        :=True
    Selection.End(xlDown).Select
    Range("A14").Select
    ActiveCell.FormulaR1C1 = "'Total"
    Range("E14").Select
    Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
    Selection.AutoFill Destination:=Range("E14:G14"), Type:=xlFillDefault
    Range("E14:G14").Select
    Rows("1:1").Font.Bold = True
    Rows("14:14").Select
    Selection.Font.Bold = True
    Cells.Select
    Selection.Columns.AutoFit
End Sub
```

Agora, no menu, selecione Executar, Executar Macro ou pressione F5. O programa executa rapidamente, mas pára antes do ponto de interrupção. O Editor do VB mostra a linha do ponto de interrupção destacada em amarelo. Você agora pode pressionar F8 para começar a analisar o código (veja Figura 2.24).

Depois de terminar a depuração do código, remova os pontos de interrupção. Você pode fazer isso clicando no ponto marrom escuro na margem para desativar o ponto de interrupção. Você também pode selecionar Depurar, Limpar Todos os Pontos de Interrupção, ou pressionar Ctrl+Shift+F9 para limpar todos os pontos de interrupção configurados no projeto.

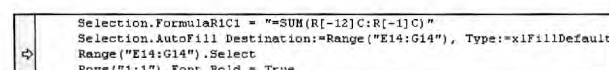
Figura 2.24
A linha amarela significa que a linha do ponto de interrupção está prestes a ser executada.

```
Sub ImportInvoice()
'
' ImportInvoice Macro
'
' Atalho de teclado: Ctrl+i
'
'    Workbooks.OpenText Filename:= _
'        "fatura.txt", Origin _
'        :=437, StartRow:=1, DataType:=xlDelimited, TextQualifier:=xlDoubleQuote _
'        , ConsecutiveDelimiter:=False, Tab:=True, Semicolon:=False, Comma:=True _
'        , Space:=False, Other:=False, FieldInfo:=Array(Array(1, 3), Array(2, 1), _
'        Array(3, 1), Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)), TrailingMinusNumbers _
'        :=True
    Selection.End(xlDown).Select
    Range("A14").Select
    ActiveCell.FormulaR1C1 = "'Total"
    Range("E14").Select
    Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
    Selection.AutoFill Destination:=Range("E14:G14"), Type:=xlFillDefault
    Range("E14:G14").Select
    Rows("1:1").Font.Bold = True
    Rows("14:14").Select
    Selection.Font.Bold = True
    Cells.Select
    Selection.Columns.AutoFit
End Sub
```

Retrocedendo ou avançando no código

Ao analisar o código, talvez você queira pular algumas linhas. Ou talvez você tenha corrigido algumas linhas do código e quer executá-las novamente. No modo Depuração, é fácil fazer isso. Meu método favorito é utilizar o mouse para pegar a seta amarela. O cursor transforma-se em um ícone, indicando que você pode passar para a próxima linha acima ou abaixo. Arraste a linha amarela até a linha que você quer executar em seguida (veja Figura 2.25). A outra opção é clicar com o cursor na linha até a qual você quer pular e então escolher Depurar, Definir Próxima Instrução.

Figura 2.25
Maneira como o cursor aparece quando a linha amarela é arrastada até uma diferente linha do código a ser executado.

```
    Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
    Selection.AutoFill Destination:=Range("E14:G14"), Type:=xlFillDefault
    Range("E14:G14").Select
    Rows("1:1").Font.Bold = True
```

Não percorrendo cada linha de *código*

Se estiver depurando o código, talvez você queira executar uma seção do código sem analisar cada linha. Isso é comum quando você entra em um loop. Talvez você queira que o VBA execute o loop 100 vezes para que você consiga analisar todas as linhas depois do loop. É muito monótono pressionar a tecla F8 centenas de vezes para investigar cada loop. Clique com o cursor na linha que você quer depurar e pressione Ctrl+F8 ou selecione Depurar, Executar até o Cursor.

Consultando qualquer coisa ao analisar o código

Ainda não falamos sobre variáveis (o programa de gravação de macros nunca grava uma variável), mas você pode consultar o valor de qualquer coisa enquanto está no modo Depurar.

Utilizando a janela Verificação Imediata

Pressione Ctrl+G para exibir a janela Verificação Imediata no Editor do VB. Enquanto a macro estiver no modo Interrupção, você pode solicitar que o Editor do VB informe a célula atualmente selecionada, o nome da planilha ativa ou o valor de qualquer variável. A Figura 2.26 mostra vários exemplos de consultas digitadas na janela Verificação Imediata.

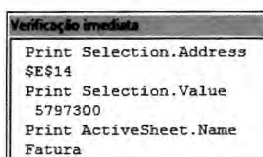
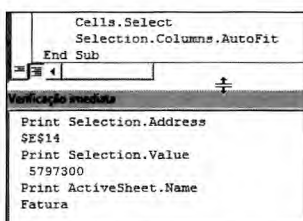
Figura 2.26
As consultas (e suas respostas) que podem ser digitadas na janela Verificação Imediata quando uma macro está no modo Interrupção.

Quando aberta com Ctrl+G, a janela Verificação Imediata normalmente aparece na parte inferior da janela código. Você pode utilizar a alça de redimensionamento (acima da barra de barra azul de título da Verificação Imediata) para diminuir ou aumentar a janela Verificação Imediata (veja Figura 2.27).

Figura 2.27
Redimensionando a janela Verificação Imediata.

Há uma barra de rolagem ao lado da janela Verificação Imediata. Você pode utilizá-la para rolar para trás ou para frente pelas entradas passadas na janela Verificação Imediata.

Não é necessário executar consultas apenas na parte inferior da janela Verificação Imediata. Eis um exemplo. Nesse caso, acabei de executar uma linha do código. Na janela Verificação Imediata, chamo `Selection.Address` para assegurar que essa linha de código funcionou (veja Figura 2.28).

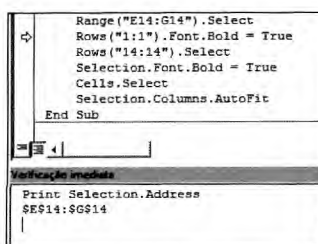
Figura 2.28
A janela Verificação Imediata mostra os resultados antes da linha atual ser executada.

Pressione a tecla F8 para executar a próxima linha do código. Em vez de redigitar a mesma consulta, clique na janela Verificação Imediata no final da linha que contém a última consulta (veja Figura 2.29).

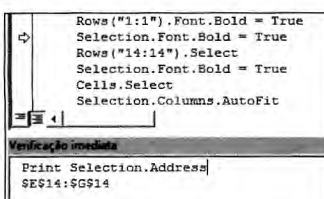
Figura 2.29
Não há necessidade de digitar os mesmos comandos na janela Verificação Imediata. Basta colocar o cursor no final do comando anterior e pressionar Enter.

Pressione Enter e a janela Verificação Imediata executará essa consulta novamente, exibindo os resultados na próxima linha e posicionando os resultados mais antigos na parte inferior da janela. Nesse caso, o endereço selecionado é `$1:$1`. A resposta anterior, `$E$14:$G$14`, é posicionada na parte inferior da janela (veja Figura 2.30).

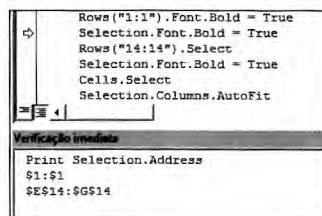
Figura 2.30
A resposta anterior (`$E$14:$G$14`) torna-se a última linha e a resposta atual (`$1:$1`) aparece abaixo da consulta.

Pressione F8 mais quatro vezes para investigar a linha do código com `Cells.Select`. Posicione novamente o cursor na janela Verificação Imediata logo após `Print Selection.Address` e pressione Enter. A consulta é executada novamente e o endereço mais recente é mostrado, com as respostas anteriores aparecendo na parte inferior da janela Verificação Imediata (veja Figura 2.31).

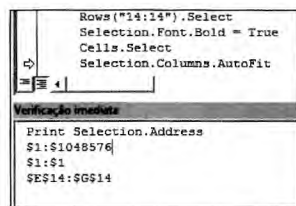
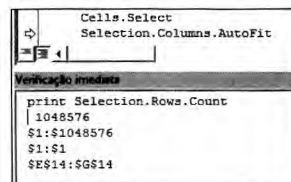
Figura 2.31
Depois de selecionar todas as células com `Cells.Select`, coloque o cursor depois da consulta na janela Verificação Imediata e pressione Enter. A nova resposta é que o intervalo selecionado são todas as linhas 1 a 1.084.576.

Também é possível utilizar esse método para alterar a consulta. Clique à direita da palavra *`Address`* na janela Verificação Imediata. Pressione a tecla backspace para apagar a palavra *`Address`* e, no lugar dela, digite **`Rows.Count`**. Pressione Enter, e a janela Verificação Imediata mostrará o número de linhas na seleção (veja Figura 2.32).

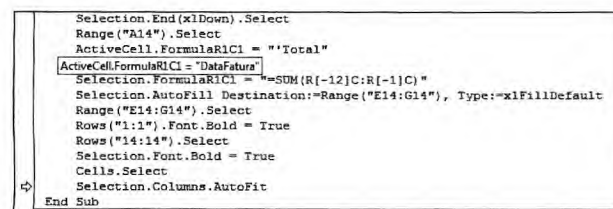
Figura 2.32
Exclua parte de uma consulta, digite algo novo e pressione Enter. As respostas anteriores serão posicionadas abaixo e a resposta atual é exibida.

Essa técnica é excelente quando você estiver tentando entender um fragmento confuso do código. Talvez eu consulte o nome da planilha ativa (`Print Activesheet.Name`), a seleção (`Print Selection.Address`), a célula ativa (`Print ActiveCell.Address`), a fórmula (`PrintActiveCell.Formula`) na célula ativa, o valor da célula ativa (`Print ActiveCell.Value` ou `Print ActiveCell`, porque `Value` é a propriedade padrão de uma célula) e assim por diante.

Para fechar a janela Verificação Imediata, clique no X no canto superior direito dessa janela. (Ctrl+G não ativa e desativa a janela).

Fazendo uma consulta com a o cursor do mouse sobre uma expressão do código

Em muitas ocasiões, você pode posicionar o cursor sobre uma expressão no código. Espere um segundo e uma Dica de Tela se abrirá, mostrando o valor atual da expressão. É muito útil que você entenda isso quando discutirmos o tópico sobre looping no Capítulo 5, "Looping e controle de fluxo". E será ainda mais útil para o código gravado. Observe que a expressão sobre a qual você posiciona o cursor não precisa estar na linha de código recém-executado. Na Figura 2.33, o Visual Basic simplesmente selecionou a Linha 1 (tornando A1 a `ActiveCell`). Se o cursor for posicionado sobre `ActiveCell.Formula`, obteremos uma Dica de Tela mostrando que a fórmula em `ActiveCell` é a palavra *`DataFatura`*.

```
    Selection.End(xlDown).Select
    Range("A14").Select
    ActiveCell.FormulaR1C1 = "'Total"
ActiveCell.FormulaR1C1 = "DataFatura"
    Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
    Selection.AutoFill Destination:=Range("E14:G14"), Type:=xlFillDefault
    Range("E14:G14").Select
    Rows("1:1").Font.Bold = True
    Rows("14:14").Select
    Selection.Font.Bold = True
    Cells.Select
    Selection.Columns.AutoFit
End Sub
```

Figura 2.33
Posicione por alguns segundos o cursor do mouse sobre qualquer expressão e uma Dica de Tela mostrará o valor atual da expressão.

Às vezes a janela VBA parece *não responder* à operação de posicionar o cursor do mouse sobre a expressão. Como algumas expressões não devem mostrar um valor, é difícil dizer se o VBA não está exibindo o valor de propósito ou se você está no modo 'não respondendo' falho (*buggy*). Tente posicionar o cursor sobre algo que você saiba que irá responder (como uma variável). Se não receber uma resposta, posicione o cursor, clique na variável e mantenha o cursor sobre ela. Isso tende a despertar o Excel do estado de letargia e esse recurso funcionará novamente.

Você continua impressionado? Comecei o capítulo reclamando que isso não era muito parecido com o BASIC, mas tenho de admitir que é ótimo trabalhar no ambiente Visual Basic. Essas ferramentas de depuração são excelentes.

Fazendo uma consulta por meio de uma janela de inspeção

No Visual Basic, um relógio não é algo que se usa no pulso. Ele permite observar o valor de qualquer expressão enquanto você investiga o código. Digamos que, no exemplo atual, fosse necessário observar o que é selecionado à medida que o código executa. Eu definiria uma inspeção para `Selection.Address`.

No menu Depurar do Editor do VB, selecione Adicionar Inspeção de Variáveis.

Na caixa de diálogo Adicionar Inspeção de Variáveis, digite **`Selection.Address`** na caixa de texto Expressão e clique em OK (veja Figura 2.34).

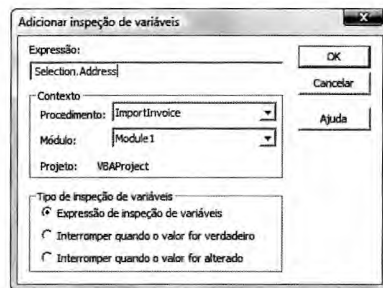
Figura 2.34
Configurando uma inspeção para ver o endereço da seleção atual.

Uma janela Inspeção de Variáveis, bastante utilizada, é adicionada à janela do Visual Basic. Em geral, ela entra na parte inferior da janela de código. Comecei executando a macro, importando o arquivo e pressionando End+Down para passar para a última linha com dados. Assim que o código `Cells.Select` é executado, a janela Inspeção de Variáveis mostra que `Selection.Address` é `$A$10` (veja Figura 2.35).

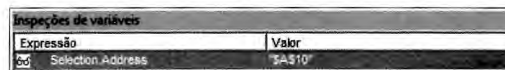
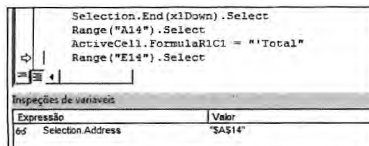
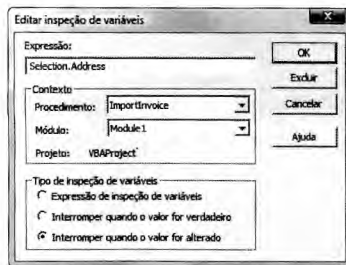
Figura 2.35
Sem usar o recurso de posicionar o cursor ou digitar na janela Verificação Imediata, você sempre poderá ver o valor de expressões inspecionadas.

Pressione a tecla F8 para executar o código `Range("A14").Select`. A janela Inspeções de Variáveis é atualizada para mostrar que o endereço atual de `Selection` é agora `$A$14` (veja Figura 2.36).

Figura 2.36
Depois de executar outra linha de código, o valor na janela Inspeções de Variáveis é atualizado para indicar o endereço da nova seleção.

Utilizando uma inspeção para definir um ponto de interrupção

Clique com o botão direito do mouse no ícone de óculos na janela Inspeções de Variáveis e escolha Editar Inspeção. Na seção Tipo de Inspeção de Variáveis da caixa de diálogo Editar Inspeção de Variáveis, selecione Interromper Quando o Valor for Alterado (veja Figura 2.37). Clique em OK.

Figura 2.37
Selecione Interromper Quando o Valor for Alterado na parte inferior da caixa de diálogo Editar Inspeção de Variáveis.

O ícone de óculos transforma-se em uma mão com um ícone de triângulo. Você agora pode pressionar F5 para executar o código. A macro começa a executar as linhas de código até que algo novo seja selecionado. Isso é muito poderoso. Em vez de investigar cada linha de código, você pode parar a macro exatamente quando algo importante acontecer. Também é possível definir que uma inspeção pare quando o valor de uma determinada variável mudar.

Utilizando uma inspeção em um objeto

No exemplo anterior, inspecionamos uma propriedade específica: `Selection.Address`. Também é possível inspecionar um objeto, como Selection. Na Figura 2.38, você definiu uma inspeção em Selection e visualiza o ícone de óculos e o ícone +.

Figura 2.38
Definir uma inspeção em um objeto posiciona o ícone + ao lado dos óculos.

Clicando no ícone +, você pode ver todas as propriedades associadas a Selection. Veja Figura 2.39! Agora você pode ver mais do que sempre quis saber sobre Selection. Existem propriedades que provavelmente você nunca percebeu que estavam disponíveis. Você pode ver que a propriedade `AddIndent` é configurada como `False` e a propriedade `AllowEdit`, como `True`. Há propriedades úteis na lista — você pode ver a `Formula` da seleção.

Figura 2.39
Clicar no ícone + mostra uma gama de propriedades e seus valores atuais.

Nessa janela Inspeção de Variáveis, pode-se expandir algumas entradas. A coleção Borders, por exemplo, tem um sinal de adição ao lado dela. Clique em qualquer ícone + para ver mais detalhes.

A última referência para todos os objetos, métodos, propriedades

No Editor do VB, pressione F2 para abrir o Pesquisador de Objeto (veja Figura 2.40). O Pesquisador de Objeto permite navegar e pesquisar toda a biblioteca de objetos do Excel. Tenho um livro grosso que é uma reimpressão desse modelo de objeto inteiro do Pesquisador de Objeto. Ele tem 409 páginas de texto. Nunca li essas 409 páginas porque o Navegador de Objeto predefinido é muito mais poderoso e sempre está disponível com um simples pressionamento da tecla F2. Reservei algumas páginas para ensinar você a utilizar o Pesquisador de Objeto.

Figura 2.40
Pressione F2 para exibir o Pesquisador de Objeto.

Pressione F2 e o Pesquisador de Objeto aparece onde a janela Códigos normalmente aparece. A lista suspensa mais em cima mostra atualmente <Todas as Bibliotecas>. Nessa lista suspensa há entradas para o Excel, o Office, o VBA e para todas as pastas de trabalho que você abriu, além de entradas adicionais para tudo o que você verifica em Ferramentas, Referências. Acesse a lista suspensa e selecione apenas Excel, por enquanto.

Na janela esquerda do Pesquisador de Objeto há uma lista de todas as classes disponíveis para o Excel. Clique na classe `Application` na janela esquerda. A janela direita se ajusta para mostrar todas as propriedades e métodos que se aplicam ao objeto Aplicativo (veja Figura 2.41). Clique em algo na janela direita, como `ActiveCell`. A janela inferior do Pesquisador de Objeto informa que `ActiveCell` é uma propriedade que retorna um intervalo. Ela informa que `ActiveCell` é de leitura — um alerta de que você não pode atribuir um endereço a `ActiveCell` para mover o ponteiro de célula.

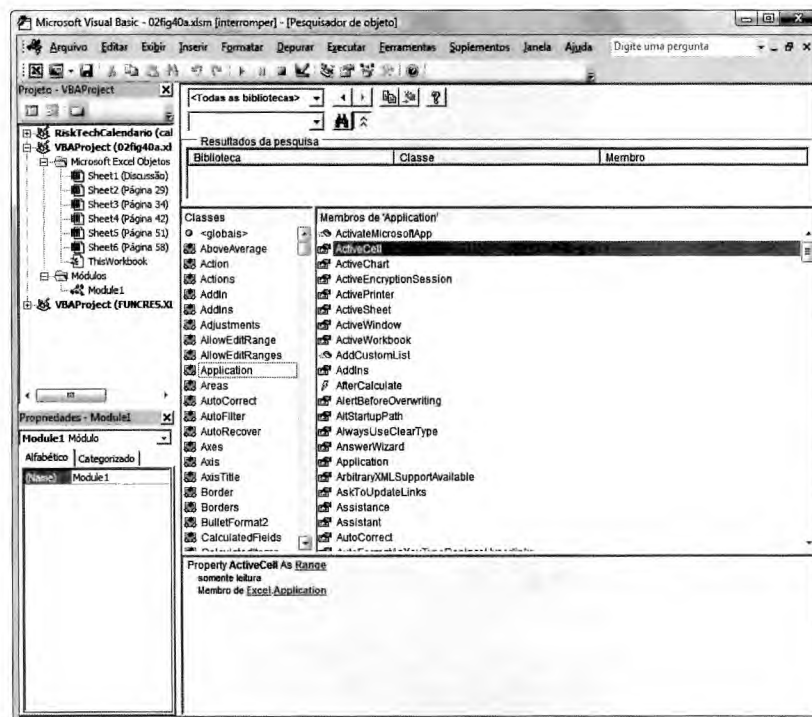
Figura 2.41
Selecione uma classe e então um membro. A janela inferior informará os princípios básicos sobre o membro selecionado. Os métodos aparecerão como livros verdes com linhas nas quais é possível mover-se rapidamente. As propriedades aparecerão como fichas de arquivo, com uma mão apontando para elas.

Selecione uma classe e então um membro. A janela inferior informa os princípios básicos sobre o membro selecionado. Os métodos aparecem como livros verdes com linhas nas quais é possível mover-se rapidamente. As propriedades aparecem como fichas de arquivo com uma mão apontando para elas.

Aprendemos no Pesquisador de Objeto que `ActiveCell` retorna um intervalo. Clique no hiperlink verde para `Range` na janela inferior e você verá todas as propriedades e métodos que são aplicados a objetos Range e, conseqüentemente, à propriedade `ActiveCell`. Clique em qualquer propriedade ou método e clique no ponto de interrogação amarelo perto da parte superior do Pesquisador de Objeto para acessar o tópico de ajuda dessa propriedade ou método.

Digite um termo qualquer na caixa de texto ao lado dos binóculos e clique nos binóculos para localizar todos os membros correspondentes da biblioteca do Excel.

As capacidades de pesquisa e hiperlinks disponíveis no Pesquisador de Objeto o tornam muito mais valioso do que uma lista impressa em ordem alfabética de todas as informações. Aprenda a usar o Pesquisador de Objeto na janela do VBA pressionando F2. Para fechar o Pesquisador de Objeto e retornar à janela Códigos, clique no X na parte inferior do canto superior direito (veja Figura 2.42).

Figura 2.42
Feche o Pesquisador de Objeto e retorne à janela Códigos clicando nesse X.

Cinco dicas fáceis para limpar o código gravado

Minhas desculpas. Estamos quase no final do Capítulo 2 e a macro absurda que gravamos no Capítulo 1 ainda não funciona. Tenho certeza de que você quer chegar ao cerne da questão e descobrir como fazer a macro gravada funcionar. Eis cinco regras fáceis que executarão rapidamente o código e farão com que ele funcione bem.

Dica 1: não selecione nada

Nada significa 'código gravado' que vai além do código que seleciona as coisas antes de agir sobre elas. Isso faz sentido — na interface do Excel, você tem de selecionar a Linha 1 antes de aplicar o negrito a ela.

Mas no VBA raramente precisamos fazer isso. Há alguns recursos falhos relacionados a gráficos em que você precisa selecionar o objeto do gráfico para que o método funcione, mas isso é uma exceção. É possível aplicar o negrito diretamente à Linha 1 sem selecioná-la. As duas linhas a seguir do código transformam-se em uma linha.

O código do programa de gravação de macros antes de ser simplificado:

```
Rows("1:1").Select
Selection.Font.Bold = True
```

Depois de simplificar o código gravado:

```
Rows("1:1").Font.Bold = True
```

Esse método tem várias vantagens. Primeiro, haverá metade do número de linhas de código no seu programa. Segundo, o programa executará mais rápido.

Depois de gravar o código, destaco literalmente antes da palavra `Select` no fim de uma linha até o ponto depois da palavra `Selection` na próxima linha e pressiono Delete (veja Figura 2.43 e 2.44).

Figura 2.43
Selecione de 'Select' até 'Selection'...

```
Range("A14").Select
ActiveCell.FormulaR1C1 = "'Total"
Range("E14").Select
Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
Selection.AutoFill Destination:=Range("E14:G14"), Type:=xlFillDefault
Range("E14:G14").Select
```

Figura 2.44
... e pressione a tecla Delete. Esse é o 'bê-a-bá' da limpeza de macros gravadas.

```
ActiveCell.FormulaR1C1 = "'Total"
Range("E14").FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
Selection.AutoFill Destination:=Range("E14:G14"), Type:=xlFillDefault
```

Dica 2: execute Range a partir da parte inferior para localizar a última linha

É difícil confiar nos dados provenientes de um lugar qualquer. Se você estiver analisando dados no Excel, lembre-se de que eles muitas vezes são provenientes de um sistema em que geralmente não conhecemos o que contém e nem há quanto tempo isso foi gravado. A verdade universal é que, cedo ou tarde, um funcionário em algum lugar encontrará uma maneira de quebrar o sistema de origem e inserir um registro sem um número de fatura. Talvez seja necessário ocorrer uma falta de energia mas, invariavelmente, não podemos contar com o fato de que cada célula esteja preenchida.

Isso é um problema ao utilizar o atalho End+Down. Essa combinação de teclas não o leva até a última linha com dados na planilha. Isso o leva até última linha com dados no intervalo atual. Na Figura 2.45, pressionar End+Down moveria o cursor para a célula A6 em vez de A12.

Figura 2.45
End+Down falha na interface com o usuário se um registro não tiver um valor. De maneira semelhante, `End(xlDown)` falha no Excel VBA.

A1 | fx | DataFatura

	A	B	C	D
1	DataFatura	NumeroFatur	NumeroVendedor	NumeroClient
2	07/06/2004	123829	S21	C8754
3	07/06/2004	123830	S45	C3390
4	07/06/2004	123831	S54	C2523
5	07/06/2004	123832	S21	C5519
6	07/06/2004	123833	S45	C3245
7		123834	S54	C7796
8	07/06/2004	123835	S21	C1654
9	07/06/2004	123836	S45	C6460
10	07/06/2004	123837	S54	C5143
11	07/06/2004	123838	S21	C7868
12	07/06/2004	123839	S45	C3310

A melhor solução é iniciar na parte inferior da planilha Excel e pressionar End+Up. Agora, isso parece ridículo na interface do Excel porque é muito fácil ver se você está realmente na linha final dos dados. Mas, como no Excel VBA é fácil iniciar na última linha, adquira o hábito de utilizar esse código para encontrar a última linha real:

```
Cells(Rows.Count, 1).End(xlUp)
```

> **NOTA**
>
> De 1995 a 2006, as planilhas do Excel apresentavam 65.536 linhas. Na edição anterior deste livro, o estilo de codificação era usar `Range("A65536").End(xlUp)` para encontrar a última linha. Com a expansão para 1,048,576 linhas, talvez você seja tentado a utilizar `Range("A1048576").End(xlUp)` no Excel 2007.
>
> Entretanto, `Rows.Count` retornará o número de linhas na pasta de trabalho ativa. Isso envolve a possibilidade de que a pasta de trabalho esteja no modo de compatibilidade ou até mesmo de que alguém esteja executando o código no Excel 2003.

Dica 3: utilizar variáveis para evitar a codificação manual de linhas e fórmulas

O programa de gravação de macros nunca grava uma variável. Elas são muito fáceis de utilizar e as discutiremos detalhadamente mais adiante; mas, assim como no BASIC, uma variável pode lembrar um valor.

Minha recomendação é definir a última linha com dados como uma variável. Gosto de utilizar nomes de variáveis descritivos, assim meu favorito nesse caso é `FinalRow`:

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
```

Agora que você conhece o número de linhas da última gravação, é fácil colocar a palavra `Total` na Coluna A da próxima linha:

```
Range("A" & FinalRow + 1).Value = "Total"
```

➔ Para métodos mais simples de referir-se a este intervalo, **consulte** "Usando a propriedade `Offset` para referenciar um intervalo," **p. 65**, no Capítulo 3, "Referenciando intervalos".

Você pode até mesmo usar a variável ao construir a fórmula. Essa fórmula soma tudo de E2 a `FinalRow` de E:

```
Range("E" & FinalRow + 1).Formula = "=SUM(E2:E" & FinalRow & ")"
```

Dica 4: aprenda a copiar e colar em uma única instrução

O código gravado é famoso por copiar um intervalo, selecionar outro e então fazer um `ActiveSheet.Paste`. O método `Copy` aplicado a um intervalo é realmente muito mais poderoso. Você pode especificar o que copiar e o destino em uma instrução.

Código gravado:

```
Range("E14").Select
Selection.Copy
Range("F14:G14").Select
ActiveSheet.Paste
```

Código melhor :

```
Range("E14").Copy Destination:=Range("F14:G14")
```

Dica 5: utilize `With...End With` se estiver executando múltiplas ações com a mesma célula ou intervalo de células

Se você fosse aplicar negrito e sublinhado duplo à linha total, com uma fonte maior e uma cor especial, talvez obtivesse um código como este:

```
Range("A14:G14").Select
Selection.Font.Bold = True
Selection.Font.Size = 12
Selection.Font.ColorIndex = 5
Selection.Font.Underline = xlUnderlineStyleDoubleAccounting
```

Para quatro dessas linhas do código, o VBA deve resolver a expressão `Selection.Font`. Como existem quatro linhas que se referem ao mesmo objeto, você pode nomeá-lo uma vez na parte superior de um bloco `With`. Dentro do bloco `With...End With`, assume-se que tudo que iniciar com um ponto refere-se ao objeto `With`:

```
With Range("A14:G14").Font
    .Bold = True
    .Size = 12
    .ColorIndex = 5
    .Underline = xlUnderlineStyleDoubleAccounting
End With
```

Juntando tudo — corrigindo o código gravado

Estudo de caso

Alterando o código gravado

Utilizando as cinco dicas dadas neste capítulo, você pode transformar o código gravado em um código eficiente, de aparência profissional. Eis o código como gravado pelo programa de gravação de macros:

```
'Macro ImportInvoice
'
'Atalhos pelo teclado: Ctrl+i
'

 Workbooks.OpenText Filename:= _
        "C:\fatura.txt", Origin _
```

```
        :=437, StartRow:=1, DataType:=xlDelimited, TextQualifier:=xlDoubleQuote _
        , ConsecutiveDelimiter:=False, Tab:=True, Semicolon:=False, Comma:=True _
        , Space:=False, Other:=False, FieldInfo:=Array(Array(1, 3), Array(2, 1), _
        Array(3, 1), Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)), _
        TrailingMinusNumbers:=True
    Selection.End(xlDown).Select
    Range("A14").Select
    ActiveCell.FormulaR1C1 = "'Total"
    Range("E14").Select
    Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
    Selection.AutoFill Destination:=Range("E14:G14"), Type:=xlFillDefault
    Range("E14:G14").Select
    Rows("1:1").Select
    Selection.Font.Bold = True
    Rows("14:14").Select
    Selection.Font.Bold = True
    Cells.Select
    Selection.Columns.AutoFit
End Sub
```

Siga estes passos para limpar a macro:

1. Não há problemas com as linhas `Workbook.OpenText` da maneira como foram gravadas.

2. As linhas seguintes do código tentam localizar a linha final dos dados, de modo que o programa saiba onde inserir a linha Total:

```
Selection.End(xlDown).Select
```

Você não precisa selecionar nada para encontrar a última linha. Isso também ajuda a atribuir o número de linha da linha final e a linha Total a uma variável para que elas possam ser utilizadas posteriormente. Para tratar o caso inesperado em que uma única célula na Coluna A está vazia, comece na parte inferior da planilha e suba até localizar a última linha utilizada:

```
' Localiza a última linha com dados. Isso pode mudar todos os dias
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
TotalRow = FinalRow + 1
```

3. Essas linhas de código inserem a palavra *Total* na Coluna A da linha Total:

```
Range("A14").Select
ActiveCell.FormulaR1C1 = "'Total"
```

O melhor código utilizará a variável `TotalRow` para localizar o local a inserir a palavra *Total*. Novamente, não há necessidade de selecionar a célula antes de inserir o rótulo:

```
' Cria uma linha Total abaixo disso
Range("A" & TotalRow).Value = "Total"
```

4. Essas linhas de código inserem a fórmula `Total` na coluna E e a copiam para as duas próximas colunas:

```
Range("E14").Select
Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
Selection.AutoFill Destination:=Range("E14:G14"), Type:=xlFillDefault
Range("E14:G14").Select
```

Não há nenhuma razão para selecionar tudo isso. A seguinte linha insere a fórmula em três células. O estilo L1C1 de fórmulas é discutido detalhadamente no Capítulo 6, "Fórmulas no estilo L1C1":

```
Range("E" & TotalRow).Resize(1, 3).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
```

5. O programa de gravação de macros seleciona um intervalo e então aplica a formatação:

```
Rows("1:1").Select
Selection.Font.Bold = True
Rows("14:14").Select
Selection.Font.Bold = True
```

Não há nenhuma razão para selecionar antes de aplicar a formatação. Essas duas linhas realizam a mesma ação e muito mais rapidamente:

```
Rows("1:1").Font.Bold = True
Rows(TotalRow & ":" & TotalRow).Font.Bold = True
```

6. O programa de gravação de macros seleciona todas as células antes de executar o comando `AutoFit`:

```
Cells.Select
Selection.Columns.AutoFit
```

Não há necessidade de selecionar as células antes de executar `AutoFit`:

```
Cells.Columns.AutoFit
```

7. O programa de gravação de macros adiciona uma breve descrição na parte superior de cada macro:

```
'Macro ImportInvoice
```

Agora que você mudou o código da macro gravada para algo que realmente funcionará, sinta-se à vontade para adicionar seu nome como o autor à descrição e mencionar a função da macro:

```
'Macro ImportInvoice
'Escrita por Bill Jelen, esta importa fatura.txt e inclui totais.
```

Eis a macro final com as alterações:

```
Sub ImportInvoiceFixed()
'
'Macro ImportInvoice
'Escrita por Bill Jelen, essa macro importa fatura.txt e inclui totais.
'
' Tecla de atalho: Ctrl+i
'
    Workbooks.OpenText Filename:= _
        "C:\fatura.txt", Origin _
        :=437, StartRow:=1, DataType:=xlDelimited, TextQualifier:=xlDoubleQuote _
        , ConsecutiveDelimiter:=False, Tab:=True, Semicolon:=False, Comma:=True _
        , Space:=False, Other:=False, FieldInfo:=Array(Array(1, 3), Array(2, 1), _
        Array(3, 1), Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)), _
        TrailingMinusNumbers:=True
    ' Localiza a última linha com dados. Isso pode mudar todos os dias
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    TotalRow = FinalRow + 1
    ' Inclui uma linha Total a seguir
    Range("A" & TotalRow).Value = "Total"
    Range("E" & TotalRow).Resize(1, 3).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
    Rows("1:1").Font.Bold = True
    Rows(TotalRow & ":" & TotalRow).Font.Bold = True
    Cells.Columns.AutoFit
End Sub
```

Próximos passos

Agora, nosso objetivo é saber como gravar uma macro. Você pode utilizar a ajuda e depuração para descobrir como o código funciona. Há também cinco ferramentas para tornar a aparência do código gravado mais profissional.

Os próximos capítulos apresentam mais detalhes a respeito de como se referir a intervalos, loops e ao absurdo, mas útil, estilo L1C1 das fórmulas que o programa de gravação de macros adora usar. Além disso, apresentamos 30 exemplos de código úteis que você pode utilizar.

Referenciando intervalos

3

NESTE CAPÍTULO

Um *intervalo* pode ser uma célula, uma linha, uma coluna ou um agrupamento disso. O objeto `RANGE` talvez seja o objeto mais utilizado no Excel VBA — afinal, você está manipulando dados em uma planilha. Embora um intervalo possa referenciar qualquer agrupamento de células em uma planilha, ele só poderá referenciar uma planilha por vez; se quiser referenciar intervalos em várias planilhas, cada planilha deve ser referenciada separadamente.

Este capítulo mostra diferentes maneiras de referenciar intervalos como, por exemplo, especificar uma linha ou uma coluna. Você também aprenderá a manipular células com base na célula ativa e a criar um novo intervalo a partir de intervalos sobrepostos.

O objeto Range

A seguir há uma hierarquia dos objetos do Excel:

Aplicativo ➔ *Pasta de trabalho* ➔ *Planilha* ➔ *Intervalo*

O objeto Range é uma propriedade do objeto Worksheet. Isso significa que o objeto Range requer que uma planilha esteja ativa ou que ele referencie uma planilha. As duas linhas a seguir significam a mesma coisa se Worksheets(1) for a planilha ativa:

```
Range("A1")
Worksheets(1).Range("A1")
```

Existem diversas maneiras de fazer referência a um objeto Range. `Range("A1")` é mais identificável, uma vez que é como faz o gravador de macro. Mas todas as formas a seguir são equivalentes:

```
Range("D5")
[D5]
Range("B3").Range("C3")
Cells(5,4)
Range("A1").Offset(4,3)
Range("MyRange") 'presumindo que D5 se chame MyRange
```

Qual formato utilizar depende das suas necessidades. Continue lendo — logo tudo isso fará sentido!

Utilizando os cantos superior esquerdo e inferior direito de uma seleção para especificar um intervalo

A propriedade `Range` tem duas sintaxes aceitáveis. Para especificar um intervalo retangular na primeira sintaxe, especificamos a referência completa do intervalo assim como especificaríamos uma fórmula no Excel:

```
Range("A1:B5").Select
```

usa dois pontos

usa Vírgula

Na segunda sintaxe, ou sintaxe alternativa, você especifica os cantos superior esquerdo e inferior direito do intervalo retangular desejado. Nessa sintaxe, a instrução equivalente poderia ser esta:

```
Range("A1", "B5").Select
```

Para qualquer canto, é possível substituir um intervalo nomeado, a propriedade `Cells` ou a propriedade `ActiveCell`. Essa linha de código seleciona o intervalo retangular de A1 até a célula ativa:

```
Range("A1", ActiveCell).Select
```

A instrução a seguir seleciona cinco linhas abaixo e duas colunas à direita a partir da célula ativa:

```
Range(ActiveCell, ActiveCell.Offset(5, 2)).Select
```

Intervalos nomeados

Você provavelmente já utilizou intervalos nomeados em planilhas e fórmulas. Também pode utilizá-los no VBA.

Para referenciar o intervalo `"MyRange"` em Sheet1, faça isto:

```
Worksheets("Sheet1").Range("MyRange")
```

Note que o nome do intervalo está entre aspas, diferentemente do uso de intervalos nomeados em fórmulas na planilha em si. Se você se esquecer de colocar o nome entre aspas, o Excel achará que você está se referindo a uma variável no programa, a menos que esteja usando a sintaxe de atalho discutida na seção anterior; nesse caso, as aspas não são empregadas.

O atalho para referenciar intervalos

Há um atalho disponível ao referenciar intervalos. Esse atalho utiliza [ ] (colchetes), como mostrado na Tabela 3.1.

Tabela 3.1 Atalhos para referenciar intervalos

Método padrão	Atalho
`Range("D5")`	`[D5]`
`Range("A1:D5")`	`[A1:D5]`
`Range ("A1:D5," "G6:I17")`	`[A1:D5, G6:I17]`
`Range("MyRange")`	`[MyRange]`

Referenciando intervalos em outras planilhas

Alternar entre planilhas ativando a planilha necessária pode tornar o código excessivamente lento. Para evitar essa lentidão, você pode referenciar uma planilha que não está ativa referenciando primeiro o objeto Worksheet:

```
Worksheets("Plan1").Range("A1")
```

Essa linha do código referencia Plan1 da pasta de trabalho ativa mesmo se Plan2 for a planilha ativa.

Se precisar referenciar um intervalo em outra pasta de trabalho, inclua os objetos Workbook, Worksheet e Range:

```
Workbooks("DataFatura.xls").Worksheets("Plan1").Range("A1")
```

Seja cuidadoso se utilizar a propriedade `Range` como um argumento dentro de outra propriedade `Range`. Você deve identificar o intervalo completamente todas as vezes. Suponha, por exemplo, que Plan1 seja a planilha ativa e você precise somar os dados a partir de Plan2:

```
WorksheetFunction.Sum(Worksheets("Plan2").Range(Range("A1"), Range("A7")))
```

Essa linha não funciona. Por quê? Porque `Range(Range("A1"), Range("A7"))` referencia um intervalo extra no início da linha de código. O Excel não assume que você quer transportar a referência de objeto Worksheet para os outros objetos Range. Portanto, o que você faz? Bem, você poderia escrever isto:

```
WorksheetFunction.Sum(Worksheets("Plan2").Range(Worksheets("Plan2").Range("A1"),
    Worksheets("Plan2").Range("A7")))
```

Mas essa não apenas é uma longa linha de código, como também é difícil de ler! Felizmente, há uma maneira mais simples, com `With...End With`:

```
With Worksheets("Plan2")
    WorksheetFunction.Sum(.Range(.Range("A1"), .Range("A7")))
End With
```

Note agora que há um `.Range` no código, mas sem a referência do objeto anterior. Isso ocorre porque `With Worksheets("Sheet2")` quer dizer que o objeto do intervalo é a planilha.

Referenciando um intervalo que é relativo a outro

Em geral, o objeto `RANGE` é uma propriedade de uma planilha. Também é possível tornar `RANGE` a propriedade de outro intervalo. Nesse caso, a propriedade `Range` é relativa ao intervalo original! Isso resulta em um código bastante não-intuitivo. Considere este exemplo.

```
Range("B5").Range("C3").Select
```

Na verdade, isso seleciona a célula D7. Pense na célula C3. Ela está localizada duas linhas abaixo e duas colunas à direita da célula A1. A linha do código anterior inicia na célula B5. Se B5 estivesse na posição A1, o VBA localizaria a célula que estivesse na posição C3 relativa à B5. Em outras palavras, o VBA localizaria a célula que estivesse duas linhas abaixo e duas colunas à direita de B5, que é D7.

Novamente, considero esse estilo de codificação bastante não-intuitivo. Essa linha de código menciona dois endereços e a célula real sendo selecionada não é nenhum deles! Parece confuso quando você tenta ler esse código.

Talvez você pense em utilizar essa sintaxe para referenciar uma célula relativa à célula ativa. Por exemplo, essa linha ativa a célula três linhas abaixo e quatro colunas à direita da célula atualmente ativa:

```
Selection.Range("E4").Select
```

Essa sintaxe é mencionada apenas porque o programa de gravação de macros a utiliza. Lembre-se do Capítulo 1, "Libere o poder do Excel com o VBA!", quando gravamos uma macro com Referências Relativas habilitadas, a linha a seguir foi gravada:

```
ActiveCell.Offset(0, 4).Range("A2").Select
```

Você encontrou a célula quatro colunas à direita da célula ativa e, a partir daí, selecionou a célula que corresponderia a A2. Essa não é a maneira mais fácil de escrever código, mas é aquela que o programa de gravação de macros utiliza.

Embora uma planilha normalmente seja o objeto da propriedade `Range`, às vezes, por exemplo, durante a gravação, um intervalo poderia ser a propriedade de um intervalo.

Utilizando a propriedade `Cells` para selecionar um intervalo

A propriedade `Cells` refere-se a todas as células do objeto especificado no intervalo, o que pode ser uma planilha ou um intervalo de células. Por exemplo, esta linha seleciona todas as células da planilha ativa:

```
Cells.Select
```

Utilizando a propriedade `Cells` com o objeto Range talvez pareça redundante:

```
Range("A1:D5").Cells
```

A linha refere-se ao objeto Range original. Mas a propriedade `Cells` tem uma propriedade, `Item`, que torna a propriedade `Cells` muito útil. A propriedade `Item` permite referenciar uma célula específica em relação ao objeto Range.

A sintaxe para utilizar a propriedade `Item` com a propriedade `Cells` é a seguinte:

```
Cells.Item(Row,Column)
```

Você deve utilizar um valor numérico para `Row`, mas pode utilizar o valor numérico ou valor de string para `Column`. Ambas as linhas a seguir referem-se à célula C5:

```
Cells.Item(5,"C")
Cells.Item(5,3)
```

Como a propriedade `Item` é a propriedade padrão do objeto `RANGE`, você pode encurtar essas linhas:

```
Cells(5,"C")
Cells(5,3)
```

A capacidade de usar valores numéricos para parâmetros mostra-se especialmente útil quando é preciso fazer um loop por linhas ou colunas. O programa de gravação de macros normalmente utiliza algo como `Range("A1"). Select` para uma única célula e `Range("A1:C5").Select` para um intervalo de células. Se estiver aprendendo a codificar a partir do gravador, talvez você fique tentado a escrever o código desta maneira:

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
For i = 1 to FinalRow
    Range("A" & i & ":E" & i).Font.Bold = True
Next i
```

Esse pequeno fragmento de código, que faz um loop pelas linhas e aplica negrito às células nas Colunas A a E, é difícil de ler e de gravar. Mas, o que mais você poderia fazer?

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
For i = 1 to FinalRow
    Cells(i,"A").Resize(,5).Font.Bold = True
Next i
```

Em vez de tentar digitar o endereço do intervalo, o novo código utiliza as propriedades `Cells` e `Resize` para localizar a célula requerida, com base na célula ativa.

Utilizando a propriedade `Cells` na propriedade `Range`

Você pode utilizar propriedades `Cells` como parâmetros na propriedade `Range`. A linha a seguir refere-se ao intervalo A1:E5:

```
Range(Cells(1,1),Cells(5,5))
```

Isso é especialmente útil quando você precisa especificar as variáveis com um parâmetro, como no exemplo do loop anterior.

Utilizando a propriedade `Offset` para referenciar um intervalo

Já vimos uma referência a `Offset`; o programa de gravação de macros utilizou-a quando gravávamos uma referência relativa. Essa referência permite manipular uma célula com base no local da célula ativa. Dessa maneira, você não precisa conhecer o endereço de uma célula.

A sintaxe da propriedade `Offset` é esta:

```
Range.Offset(RowOffset, ColumnOffset)
```

Para afetar a célula F5 a partir da célula A1, escreva isto:

```
Range("A1").Offset(RowOffset:=4, ColumnOffset:=5)
```

Ou, com uma sintaxe ainda menor, escreva isto:

```
Range("A1").Offset(4,5)
```

A contagem inicia em A1, mas não inclui A1.

E se você só precisar verificar uma linha ou coluna, mas não ambas? Não será necessário inserir os parâmetros de linha e coluna. Se precisar referenciar uma célula ou uma coluna à direita ou à esquerda, utilize um destes parâmetros:

```
Range("A1").Offset(ColumnOffset:=1)
Range("A1").Offset(,1)
```

As duas linhas significam a mesma coisa. A escolha é a sua. Referenciar uma célula uma linha acima é semelhante:

```
Range("B2").Offset(RowOffset:=-1)
Range("B2").Offset(-1)
```

Mais uma vez, a escolha é sua. É uma questão de legibilidade do código.

Vamos supor que você tenha uma lista de produtos com os totais aparecendo ao lado deles e queira localizar qualquer total igual a zero e colocar BAIXO na célula ao lado dele. Você poderia fazer isso desta forma:

```
Set Rng = Range("B1:B16").Find(What:="0", LookAt:=xlWhole, LookIn:=xlValues)
Rng.Offset(, 1).Value = "BAIXO"

Sub MyOffset()
With Range("B1:B16")
    Set Rng = .Find(What:="0", LookAt:=xlWhole, LookIn:=xlValues)
    If Not Rng Is Nothing Then
        firstAddress = Rng.Address
```

```
        Do
            Rng.Offset(, 1).Value = "BAIXO"
            Set Rng = .FindNext(Rng)
        Loop While Not Rng Is Nothing And Rng.Address <> firstAddress
    End If
End With
End Sub
```

Os totais BAIXO são anotados rapidamente pelo programa, como mostrado na Figura 3.1.

O deslocamento não é só para células únicas — ele pode ser usado com intervalos. Você pode mudar novamente o foco de um intervalo da mesma maneira como pode mudar a célula ativa. A linha a seguir referencia B2:D4 (veja Figura 3.2):

```
Range("A1:C3").Offset(1,1)
```

Figura 3.1
Localize o produto com o total 0.

	A	B	C
1	Maçãs	45	
2	Laranjas	12	
3	Uva	86	
4	Limões	0	BAIXO
5	Tomates	58	

Figura 3.2
Deslocando um intervalo — `Range("A1:C3").Offset(1,1).Select`.

Utilizando a propriedade `Resize` para alterar o tamanho de um intervalo

A propriedade `Resize` permite alterar o tamanho de um intervalo com base na localização da célula ativa. Você pode criar um novo intervalo conforme suas necessidades.

A sintaxe da propriedade `Resize` é esta:

```
Range.Resize(RowSize, ColumnSize)
```

Para criar um intervalo B3:D13, utilize isto:

```
Range("B3").Resize(RowSize:=11, ColumnSize:=3)
```

Ou, mais simples, utilize isto:

```
Range("B3").Resize(11, 3)
```

E se precisar redimensionar apenas uma linha ou uma coluna e não ambas? Não é preciso inserir os parâmetros de linha e de coluna. Se precisar expandir duas colunas, utilize um destes parâmetros:

```
Range("B3").Resize(ColumnSize:=2)
```

ou

```
Range("B3").ReSize(,2)
```

As duas linhas significam a mesma coisa. A escolha é a sua. Redimensionar apenas as linhas é semelhante a:

```
Range("B3").Resize(RowSize:=2)
```

ou

```
Range("B3").Resize(2)
```

Mais uma vez, a escolha é sua. É uma questão de legibilidade do código.

Na lista de produtos, localize o total zero e aplique cor às células do total e produto correspondente (veja Figura 3.3):

```
Set Rng = Range("B1:B16").Find(What:="0", LookAt:=xlWhole, LookIn:=xlValues)
Rng.Offset(, -1).Resize(, 2).Interior.ColorIndex = 15
```

Note que a propriedade `Offset` foi utilizada primeiro para mover a célula ativa; quando você está redimensionando, a célula do canto superior esquerdo deve permanecer a mesma.

O redimensionamento não é apenas para células únicas — ele também pode ser utilizado para redimensionar um intervalo existente. Por exemplo, se você tiver um intervalo nomeado e precisar dele e das duas colunas ao lado, utilize isto:

```
Range("Hortifruti").Resize(,2)
```

Lembre-se de que o número que você redimensiona é o número total de linhas/colunas que você quer incluir.

Figura 3.3
Redimensionando um intervalo para estender a seleção.

	A	B	C
1	Maçãs	45	
2	Laranjas	12	
3	Uva	86	
4	Limões	0	
5	Tomates	58	

Utilizando as propriedades `Columns` e `Rows` para especificar um intervalo

`Columns` e `Rows` referem-se às colunas e linhas de um objeto Range especificado, que pode ser uma planilha ou um intervalo de células. Essas propriedades retornam um objeto Range que referencia as linhas ou colunas do objeto especificado.

Já vimos a linha a seguir ser utilizada, mas o que ela faz?

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
```

Essa linha de código localiza a última linha em uma planilha em que a Coluna A tem um valor e adiciona o número de linha desse objeto Range a `FinalRow`. Isso pode ser muito útil quando você precisa fazer um loop por uma planilha linha a *linha — você saberá* exatamente quantas linhas precisa analisar.

ATENÇÃO

Algumas propriedades de colunas e linhas requerem linhas e colunas contíguas para que funcionem adequadamente. Por exemplo, se você fosse utilizar a linha a seguir do código, a resposta seria 9, porque somente o primeiro intervalo seria avaliado:

```
Range("A1:B9, C10:D19").Rows.Count
```

Mas se os intervalos estivessem agrupados separadamente,

```
Range("A1:B9", "C10:D19").Rows.Count
```

a resposta seria 19.

Utilizando o método `Union` para unir vários intervalos

O método `Union` permite unir dois ou mais intervalos não-contíguos. Esse método cria um objeto temporário dos múltiplos intervalos, permitindo que você os manipule conjuntamente:

```
Application.Union(argument1, argument2, etc.)
```

O código a seguir une dois intervalos nomeados na planilha, insere a fórmula `=RAND()` e aplica negrito:

```
Set UnionRange = Union(Range("Range1"), Range("Range2"))
With UnionRange
    .Formula = "=RAND()"
    .Font.Bold = True
End With
```

Utilizando o método `Intersect` para criar um novo intervalo de intervalos sobrepostos

O método `Intersect` retorna as células que se sobrepõem entre dois ou mais intervalos:

```
Application.Intersect(argument1, argument2, etc.)
```

O código a seguir aplica cor às células sobrepostas dos dois intervalos.

```
Set IntersectRange = Intersect(Range("Range1"), Range("Range2"))
IntersectRange.Interior.ColorIndex = 6
```

Utilizando a função `ISEMPTY` para verificar se uma célula está vazia

A função `ISEMPTY` retorna um valor Booleano se uma única célula estiver ou não vazia —`True` se vazia; `False`, caso contrário. A célula deve estar realmente vazia. Mesmo se ela tiver um espaço que não possa ser visto, o Excel não a considerará vazia:

```
IsEmpty(Cell)
```

Veja a Figura 3.4. Há vários grupos de dados separados por uma linha em branco. É recomendável tornar as separações um pouco mais óbvias.

Figura 3.4
Linhas em branco vazias separando dados.

	A	B	C	D
1	Maçãs	Laranjas	Uva	Limões
2	45	43	18	27
3	92%	10%	50%	6%
4				
5	Tomates	Repolho	Alface	Pimentão Verde
6	4	33	34	24
7	71%	78%	87%	72%
8				
9	Batatas	Inhame	Cebola	Alho
10	64	67	32	71
11	59%	95%	66%	84%

O código a seguir é posicionado abaixo dos dados na Coluna A; onde ele encontrar uma célula vazia, aplicará cor nas primeiras quatro células dessa linha (veja Figura 3.5):

```
LastRow = Cells(Rows.Count, 1).End(xlUp).Row
For i = 1 To LastRow
    If IsEmpty(Cells(i, 1)) Then
        Cells(i, 1).Resize(1, 4).Interior.ColorIndex = 1
    End If
Next i
```

Figura 3.5
Linhas coloridas separando dados.

	A	B	C	D
1	Maçãs	Laranjas	Uva	Limões
2	45	12	86	15
3	54%	74%	68%	83%
4				
5	Tomates	Repolho	Alface	Pimentão Verde
6	58	24	31	0
7	9%	59%	40%	77%
8				
9	Batatas	Inhame	Cebola	Alho
10	10	61	26	29
11	91%	28%	83%	37%

Utilizando a propriedade `CurrentRegion` para selecionar rapidamente um intervalo de dados

`CurrentRegion` retorna um objeto Range que representa um conjunto de dados contíguos. Contanto que os dados estejam entre uma linha vazia e uma coluna vazia, é possível selecionar a tabela com `CurrentRegion`:

```
RangeObject.CurrentRegion
```

Veja Figura 3.6. A linha a seguir seleciona A1:D3 porque esse é intervalo contíguo das células em torno da célula A1:

```
Range("A1").CurrentRegion.Select
```

Isso é útil se você tiver uma tabela cujo tamanho está em fluxo constante.

Figura 3.6
Utilize `CurrentRegion` para selecionar rapidamente um intervalo de dados contíguos em torno da célula ativa.

	A	B	C	D
1	Maçãs	Laranjas	Uva	Limões
2	82	80	46	59
3	17%	78%	2%	46%
4				

Estudo de caso

Utilizando o método `SpecialCells` para selecionar células específicas

Mesmo os usuários avançados do Excel talvez ainda não tenham encontrado a caixa de diálogo Ir para especial. Se você pressionar a tecla F5 em uma planilha Excel, a caixa de diálogo Ir para normal aparecerá na tela (veja Figura 3.7). No canto inferior esquerdo dessa caixa de diálogo há um botão chamado Especial. Clique nele para abrir a superpoderosa caixa de diálogo Ir para especial (veja Figura 3.8).

Na interface do Excel, a caixa de diálogo Ir para especial permite selecionar apenas células com fórmulas, ou apenas células em branco, ou apenas células visíveis. Selecionar somente células visíveis é excelente para entender os resultados visíveis dos dados AutoFiltrados.

Para simular a caixa de diálogo Ir Para Especial no VBA, utilize o método `SpecialCells`. Isso permite que você atue nas células que atendem certos critérios:

```
RangeObject.SpecialCells(Type, Value)
```

Figura 3.7
Embora a caixa de diálogo Ir Para não pareça muito útil, clique no botão Especial no canto esquerdo inferior.

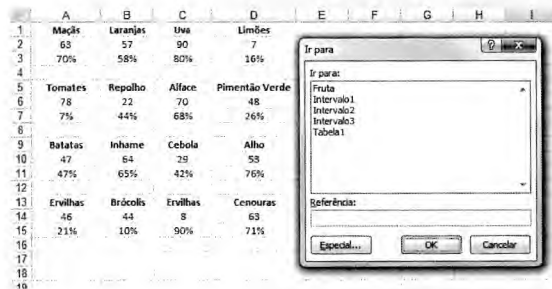
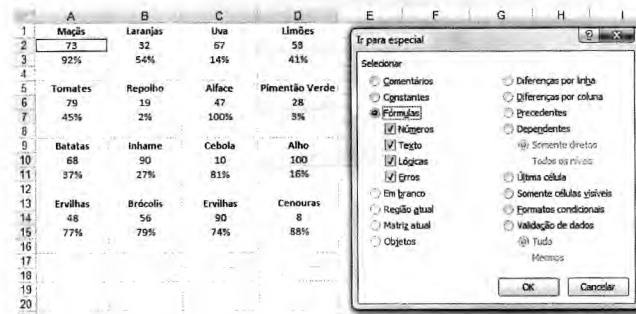



Figura 3.8
A caixa de diálogo Ir Para Especial tem muitas ferramentas de seleção incrivelmente úteis.

Esse método tem dois parâmetros: `Type` e `Value`. `Type` é um das constantes `xlCellType`:

```
xlCellTypeAllFormatConditions
xlCellTypeAllValidation
xlCellTypeBlanks
xlCellTypeComments
xlCellTypeConstants
xlCellTypeFormulas
xlCellTypeLastCell
xlCellTypeSameFormatConditions
xlCellTypeSameValidation
xlCellTypeVisible
```

`Value` é opcional e pode ser uma das seguintes:

```
xlErrors
xlLogical
xlNumbers
xlTextValues
```

O próximo código retorna todos os intervalos que foram definidos com a formatação condicional. Esse código produz um erro se não houver nenhuma formatação condicional. Ele adiciona uma borda em torno de cada seção contígua que localizar:

```
Set rngCond = ActiveSheet.Cells.SpecialCells(xlCellTypeAllFormatConditions)
If Not rngCond Is Nothing Then
    rngCond.BorderAround xlContinuous
End If
```

Alguém já lhe enviou uma planilha sem nenhum rótulo preenchido? Algumas pessoas acham que os dados mostrados na Figura 3.9 parecem perfeitos. Eles são inseridos no campo Região apenas uma vez para cada região. Isso poderia parecer esteticamente agradável, mas é impossível de classificar. Mesmo a tabela dinâmica do Excel sempre retorna dados nesse formato irritante.

Figura 3.9
As células em branco na coluna Região tornam bem difícil classificar tabelas de dados como essa.

	A	B	C
1	Região	Produto	Vendas
2	Norte	ABC	766.469
3		DEF	776.996
4		XYZ	832.414
5	Leste	ABC	703.255
6		DEF	891.799
7		XYZ	897.949
8	Oeste	ABC	631.646
9		DEF	494.919
10		XYZ	712.365
11			

Utilizar o método `SpecialCells` para selecionar todas as lacunas nesse intervalo é uma maneira rápida de preencher todas células de região vazias com a região localizada acima delas:

```
Sub FillIn()
    On Error Resume Next 'Isso é necessário porque se não houver células vazias,
    'o código falhará
    Range("A1").CurrentRegion.SpecialCells(xlCellTypeBlanks).FormulaR1C1 _
    = "=R[-1]C"
    Range("A1").CurrentRegion.Value = Range("A1").CurrentRegion.Value
End Sub
```

Nesse código, `Range("A1").CurrentRegion` refere-se ao intervalo contíguo dos dados no relatório. O método `SpecialCells` retorna apenas as células em branco nesse intervalo. Embora você possa ler mais sobre fórmulas no estilo L1C1 no Capítulo 6, "Fórmulas no estilo L1C1", essa fórmula particular preenche todas as células em branco com uma fórmula que aponta para a célula acima da célula em branco. A segunda linha do código é uma maneira rápida de simular o uso do comando Copiar e, depois, Colar Valores Especiais. A Figura 3.10 mostra os resultados.

Figura 3.10
Depois de a macro executar, as células em branco na coluna Região foram preenchidas com dados.

	A	B	C
1	Região	Produto	Vendas
2	Central	ABC	766.469
3	Central	DEF	776.996
4	Central	XYZ	832.414
5	Leste	ABC	703.255
6	Leste	DEF	891.799
7	Leste	XYZ	897.949
8	Oeste	ABC	631.646
9	Oeste	DEF	494.919
10	Oeste	XYZ	712.365
11			

Utilizando a coleção Áreas para retornar um intervalo não-contíguo

A coleção Áreas é uma coleção em intervalos não-contíguos dentro de uma seleção. Ela consiste em objetos Range individuais que representam intervalos contíguos das células dentro da seleção. Se esta contiver apenas uma área, a coleção Áreas conterá um único objeto Range que corresponderá a essa seleção.

Pode ser tentador fazer um loop pela planilha, copiar uma linha e colá-la em outra seção. Mas há uma maneira mais fácil (veja Figura 3.11):

```
Range("A:D").SpecialCells(xlCellTypeConstants, 1).Copy Range("I1")
```

Figura 3.11
A coleção Áreas facilita a manipulação de intervalos não-contíguos.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Maçãs	Laranjas	Uva	Limões					45	12	86	15
2	45	12	86	15					58	24	31	0
3	10%	11%	3%	18%					10	61	26	29
4									46	64	79	95
5	Tomates	Repolho	Alface	Pimentão Verde								
6	58	24	31	0								
7	77%	41%	91%	53%								
8												
9	Batatas	Inhame	Cebola	Alho								
10	10	61	26	29								
11	66%	7%	50%	83%								
12												
13	Ervilhas	Brócolis	Ervilhas	Cenouras								
14	46	64	79	95								
15	18%	19%	13%	68%								
16												

Referenciando tabelas

NOVO Com o Excel 2007, há a uma nova maneira de interagir com intervalos de dados: tabelas. Esses intervalos especiais oferecem a conveniência de referenciar intervalos nomeados, mas não são criados da mesma *maneira*. Para mais informações sobre como criar uma tabela nomeada, veja o Capítulo 8, "Crie e manipule nomes no VBA".

A própria tabela é referenciada utilizando o método padrão de referenciar um intervalo. Para referenciar dados na tabelaTabela1 em Plan1, faça isto:

```
Worksheets(1).Range("Tabela1")
```

Isso referencia apenas a parte dos dados da tabela sem incluir o cabeçalho nem a linha total. Para inclui-los, faça isto:

```
Worksheets(1).Range("Tabela1[#All]")
```

O que realmente gosto nesse novo recurso é a facilidade de referenciar colunas específicas de uma tabela. Você não precisa conhecer a quantidade de colunas em uma posição inicial nem a letra/número da coluna, assim como não precisa utilizar uma função `FIND`. Você pode simplesmente usar o nome de cabeçalho da coluna. A fim de referenciar a coluna Qtd da tabela, por exemplo, você pode fazer o seguinte:

```
Worksheets(1).Range("Tabela1[Qtd]")
```

Próximos passos

Agora que você está começando a ter uma idéia de como o Excel funciona, é hora de aplicar isso a situações úteis. O próximo capítulo examina as funções definidas pelo usuário, emprega as habilidades que você aprendeu até agora e apresenta outros métodos de programação sobre os quais aprenderá mais ao longo do livro.

Funções definidas pelo usuário

4

NESTE CAPÍTULO

Criando funções definidas pelo usuário

O Excel fornece muitas fórmulas predefinidas, mas às vezes você precisa de uma personalizada complexa que não foi oferecida — por exemplo, uma que some um intervalo de células com base em sua cor interna.

Então, o que você faz? Você poderia rolar pela lista e copiar as células coloridas para outra seção. Ou talvez você tenha uma calculadora ao lado enquanto rola pela lista — cuidado para não inserir o mesmo número duas vezes! Ambos os métodos são demorados e propensos a acidentes. O que fazer? Você poderia escrever um procedimento — afinal, este livro é sobre isso. Mas, há outra opção: funções definidas pelo usuário (*user-defined functions* – UDFs).

Você pode criar funções no VBA que podem ser utilizadas exatamente como as funções predefinidas do Excel, como SOMA. Depois que a função personalizada é criada, o usuário só precisa conhecer o nome dela e seus argumentos.

> NOTA
>
> As funções definidas pelo usuário (FDUs) podem ser inseridas apenas em módulos padrão. Os módulos Sheet e ThisWorkbook são um tipo de módulo especial; se você inserir a função neles, o Excel não reconhecerá que você está criando uma FDU.

Estudo de caso

Funções personalizadas — exemplo e explicação

Vamos construir uma função personalizada para adicionar dois valores. Depois de ser criada, essa função será utilizada em uma planilha.

Insira um novo módulo no Editor do VB. Digite a seguinte função no módulo. Ela é uma função chamada SOMA, que somará dois números em células diferentes. A função tem dois argumentos:

```
Add(Number1,Number2)
```

`Number1` é o primeiro número a adicionar; `Number2` é o segundo:

```
Function Add(Number1 As Integer, Number2 As Integer) As Integer
Add = Number1 + Number2
End Function
```

Vamos dividir isso:

- Nome da função: ADD.
- Os argumentos são colocados entre parênteses depois do nome da função. Esse exemplo tem dois argumentos: `Número1` e `Número2`.
- `As Integer` define o tipo de variável do resultado como um número inteiro.
- `ADD =Number1+Number2`: o resultado da função é retornado.

Eis como utilizar a função em uma planilha:

1. Digite os números nas células A1 e A2.

2. Selecione a célula A3.
3. Pressione Shift+F3 para abrir a caixa de diálogo Colar Função (ou na faixa de Opões Fórmulas, escolha Inserir Função).
4. *Selecione a categoria* Definida pelo Usuário (veja Figura 4.1).

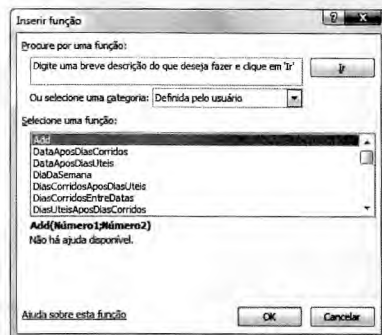
Figura 4.1
Você pode localizar suas FDUs sob a categoria Definida pelo Usuário na caixa de diálogo Inserir Função.

5. Selecione a função SOMA().
6. Na primeira caixa de argumentos, selecione a célula A1 (veja Figura 4.2).

Figura 4.2
Utilize a caixa de diálogo Argumentos da Função para inserir seus argumentos.

7. Na segunda caixa de argumentos, selecione célula A2.
8. Clique em OK.

   Parabéns! Você criou sua primeira função personalizada.

NOTA Você pode compartilhar facilmente funções personalizadas porque os usuários não precisam saber como a função funciona. Veja a seção "Compartilhando FDUs" mais adiante neste capítulo para informações adicionais.

A maioria das mesmas funções usadas em planilhas também pode ser utilizada no VBA e vice-versa. No VBA, porém, você chama a FDU (SOMA) a partir de um procedimento (Adição):

```
Sub Addition ()
Dim Total as Integer
Total = Add (1,10) 'usamos uma função Add definida pelo usuário
MsgBox "A resposta é: " & Total
End Sub
```

Compartilhando FDUs

O local em que você armazena uma FDU afeta a maneira como pode compartilhá-la:

- **Pessoal.xlsb** — Se a FDU for apenas para uso próprio e não se pretender utilizá-la em uma pasta de trabalho aberta em outro computador, você poderá armazená-la em Pessoal.xlsb.
- **Pasta de trabalho** — Se for necessário distribuir a FDU para muitas pessoas, você poderá armazená-la na pasta de trabalho em que ela será usada.

- **Suplemento** — Se a pasta de trabalho *for* compartilhada entre um grupo seleto de pessoas, você poderá distribuí-la via um suplemento (consulte o Capítulo 27, "Criando suplementos", para obter informações sobre como criar um suplemento).
- **Modelo** — Se for necessário criar várias pastas de trabalho utilizando a FDU e se as pastas de trabalho forem distribuídas para muitas pessoas, você poderá armazená-la em um modelo.

Funções personalizadas úteis do Excel

As seções a seguir incluem uma amostragem das funções que podem ser úteis no mundo cotidiano do Excel.

Este capítulo contém funções doadas por vários programadores do Excel. Eles as consideram úteis e esperam que vocês também possam aproveitá-las.

Diferentes programadores têm diferentes estilos de programação e não reescrevemos as sugestões enviadas. À medida que revisa as linhas de código, talvez você perceba diferentes maneiras de fazer a mesma tarefa; por exemplo, a forma de se referir a intervalos.

Configure o nome da pasta de trabalho atual em uma célula

A seguinte função é utilizada para configurar o nome da pasta de trabalho ativa em uma célula, como mostrado na Figura 4.3:

```
MyName()
```

Figura 4.3
Utilize uma FDU para mostrar o nome de arquivo ou o nome de arquivo com o caminho do diretório.

Nenhum argumento é utilizado com esta função:

```
Function MyName() As String
    MyName = ThisWorkbook.Name
End Function
```

Configure o nome e o caminho de arquivo da pasta de trabalho atual em uma célula

Uma variação da função anterior que configura o caminho de arquivo e o nome da pasta de trabalho ativa em uma célula, como mostrado na Figura 4.3:

```
MyFullName()
```

Nenhum argumento é utilizado com esta função:

```
Function MyFullName() As String
    MyFullName = ThisWorkbook.FullName
End Function
```

Verifique se há uma pasta de trabalho aberta

Às vezes, é necessário verificar se há uma pasta de trabalho aberta. A função a seguir retorna `True` se a pasta de trabalho estiver aberta e `False`, caso contrário:

```
BookOpen(Bk)
```

O argumento é `Bk`, o nome da pasta de trabalho que está sendo verificada:

```
Function BookOpen(Bk As String) As Boolean
Dim T As Excel.Workbook
Err.Clear 'elimina quaisquer erros
On Error Resume Next 'se o código gerar um erro, ele é ignorado e o fluxo continua
Set T = Application.Workbooks(Bk)
BookOpen = Not T Is Nothing
'Se a pasta de trabalho estiver aberta, então T armazenará o objeto pasta de trabalho e assim
'NÃO será nada
Err.Clear
On Error GoTo 0
End Function
```

Eis um exemplo de como usar a função:

```
Sub OpenAWorkbook()
Dim IsOpen As Boolean
Dim BookName As String
BookName = "ProjectFilesChapter04.xlsm"
IsOpen = BookOpen(BookName) 'chamando a nossa função - não esquecer o parâmetro
If IsOpen Then
    MsgBox BookName & " já está aberto!"
Else
    Workbooks.Open (BookName)
End If
End Sub
```

Verifique se existe uma planilha em uma pasta de trabalho aberta

Essa função requer que a(s) pasta(s) de trabalho que ela verifica estejam abertas. A função retorna `True` se a planilha for encontrada e `False` se não for:

```
SheetExists(SName, WBName)
```

Os argumentos são estes:

`SName` — O nome da planilha que estiver sendo pesquisada

`WBName` — (Opcional) O nome da pasta de trabalho que contém a planilha

```
Function SheetExists(SName As String, Optional WB As Workbook) As Boolean
    Dim WS As Worksheet
    ' Utilize a pasta de trabalho ativa por padrão
    If WB Is Nothing Then
        Set WB = ActiveWorkbook
    End If

    On Error Resume Next
        SheetExists = CBool(Not WB.Sheets(SName) Is Nothing)
    On Error GoTo 0

End Function
```

Eis um exemplo de como utilizar essa função:

```
Sub CheckForSheet()
Dim ShtExists As Boolean
ShtExists = SheetExists("Plan9")
'note que somente um parâmetro foi passado; o nome da pasta de trabalho é opcional
If ShtExists Then
    MsgBox "A planilha existe!"
Else
    MsgBox "A planilha NÃO existe!"
End If
End Sub
```

Conte o número de pastas de trabalho em um diretório

Essa função pesquisa o diretório atual e suas subpastas, se você quiser, contando todos os arquivos de pasta de trabalho de macro do Excel (XLSM) ou apenas aqueles que iniciem com uma string de letras:

```
NumFilesInCurDir (LikeText, Subfolders)
```

Os argumentos são estes:

`LikeText` — (Opcional) Um valor de string a ser pesquisado, necessário incluir um asterisco (*); por exemplo: `Mr*`

`Subfolders` — (Opcional) `True` para pesquisar subpastas, `False` (padrão) para não pesquisar

NOTA `FileSystemObject` requer a biblioteca de referência Microsoft Scripting Runtime. Para ativar essa opção, vá para Ferramentas, Referências e marque Microsoft Scripting Runtime.

```
Function NumFilesInCurDir(Optional strInclude As String = "", _
Optional blnSubDirs As Boolean = False)
Dim fso As FileSystemObject
Dim fld As Folder
Dim fil As File
Dim subfld As Folder
Dim intFileCount As Integer
Dim strExtension As String
  strExtension = "XLSM"
  Set fso = New FileSystemObject
  Set fld = fso.GetFolder(ThisWorkbook.Path)
  For Each fil In fld.Files
    If UCase(fil.Name) Like "*" & UCase(strInclude) & "*." & _
        UCase(strExtension) Then
      intFileCount = intFileCount + 1
    End If
  Next fil
  If blnSubDirs Then
    For Each subfld In fld.Subfolders
      intFileCount = intFileCount + NumFilesInCurDir(strInclude, True)
    Next subfld
  End If
  NumFilesInCurDir = intFileCount
  Set fso = Nothing
End Function
```

Eis um exemplo de como utilizar essa função:

```
Sub CountMyWkbks()
Dim MyFiles As Integer
MyFiles = NumFilesInCurDir("MrE*", True)
MsgBox MyFiles & " arquivos(s) encontrado(s)"
End Sub
```

Recupere USERID

Alguma vez você precisou manter um registro de quem salva alterações em uma pasta de trabalho? Com a função `USERID`, você pode recuperar o nome do usuário que efetuou o login em um computador. Combine-a com a função discutida na seção "Recupere a data e a hora permanentes" e você terá um excelente arquivo de log. Você também pode utilizá-la para configurar direitos do usuário para uma pasta de trabalho:

```
WinUserName()
```

Nenhum argumento é utilizado com essa função.

> **NOTA** Essa é uma função avançada que utiliza a interface de programas aplicativos (*applications programming interface* – API), que revisaremos no Capítulo 24, "Interface de programas aplicativos (API) do Windows".

Essa primeira seção (declarações `Private`) deve estar na parte superior do módulo:

```
Private Declare Function WNetGetUser Lib "mpr.dll" Alias "WNetGetUserA" _
    (ByVal lpName As String, ByVal lpUserName As String, _
lpnLength As Long) As Long
Private Const NO_ERROR = 0
Private Const ERROR_NOT_CONNECTED = 2250&
Private Const ERROR_MORE_DATA = 234
Private Const ERROR_NO_NETWORK = 1222&
Private Const ERROR_EXTENDED_ERROR = 1208&
Private Const ERROR_NO_NET_OR_BAD_PATH = 1203&
```

Você pode colocar a seção de código a seguir em qualquer lugar do módulo, contanto que ela esteja abaixo da seção anterior:

```
Function WinUsername() As String
    'variáveis
    Dim strBuf As String, lngUser As Long, strUn As String
    'limpa do buffer o nome de usuário a partir da api func
    strBuf = Space$(255)
    'usa a api func WNetGetUser para atribuir um valor de usuário a lngUser
```

```
    'haverá uma grande quantidade de espaços em branco
    lngUser = WNetGetUser("", strBuf, 255)
    'se não houver nenhum erro a partir da chamada de função
    If lngUser = NO_ERROR Then
        'limpa o espaço de branco em strBuf e atribui val à função
        strUn = Left(strBuf, InStr(strBuf, vbNullChar) - 1)
        WinUsername = strUn
    Else
    'erro, desistir
        WinUsername = "Error :" & lngUser
    End If
End Function
```

Exemplo de função:

```
Sub CheckUserRights()
Dim UserName As String
UserName = WinUsername
Select Case UserName
    Case "Administrator"
        MsgBox "Direitos ilimitados"
    Case "Guest"
        MsgBox "Você não pode fazer alterações"
    Case Else
        MsgBox "Direitos limitados"
End Select
End Sub
```

Recupere a data e hora da última vez em que algo foi salvo

Essa função recupera a data e a hora em que algo foi salvo em qualquer pasta de trabalho, incluindo a atual, como mostrado na Figura 4.4.

Figura 4.4
Recupere a data e a hora da última vez em que algo foi salvo.

> **NOTA** A célula deve ser formatada adequadamente para exibir a data/hora.

```
LastSaved(FullPath)
```

O argumento é `FullPath`, uma string que mostra o caminho completo e o nome de arquivo do arquivo em questão:

```
Function LastSaved(FullPath As String) As Date
LastSaved = FileDateTime(FullPath)
End Function
```

Recupere data e hora permanentes

Por causa da volatilidade da função `NOW`, ela não é muito útil para marcar uma planilha com a data de criação ou edição — toda vez que a pasta de trabalho é aberta ou recalculada, o resultado da função `NOW` é atualizado.

A seguinte função utiliza a função `NOW`; mas como você precisa reinserir a célula para atualizar a função, ela torna-se bem menos volátil, como mostrado na Figura 4.5:

```
DateTime()
```

Figura 4.5
Recupere a data e a hora permanentes.

Nenhum argumento é utilizado com esta função:

```
DateTime()
```

> **NOTA** A célula deve ser formatada adequadamente para exibir a data/hora.

Exemplo de função:

```
Function DateTime()
    DateTime = Now
End Function
```

Valide um endereço de e-mail

Se gerenciar uma lista de assinaturas de e-mail, você poderá receber endereços de e-mail inválidos, como endereços com um espaço antes do símbolo @. A função `ISEMAILVALID` pode verificar endereços e confirmar se eles são endereços adequados de e-mail (veja Figura 4.6):

Figura 4.6
Validando endereços de e-mail.

	A	B	C	D
1	Tracy@ MrExcel.com	FALSO	<-um espaço depois de @	
2	Bill@MrExcel.com	VERDADEIRO		
3	consult?@MrExcel/com	FALSO	<-Caracteres inválidos	
4				

ATENÇÃO

Essa função não pode verificar se um endereço de e-mail é um endereço existente. Ela verifica a sintaxe para confirmar se o endereço é legítimo.

```
IsEmailValid (StrEmail)
```

O argumento é `StrEmail`, um endereço de e-mail:

```
Function IsEmailValid(strEmail As String) As Boolean
Dim strArray As Variant
Dim strItem As Variant
Dim i As Long
Dim c As String
Dim blnIsItValid As Boolean
blnIsItValid = True
'conta o @ na string
i = Len(strEmail) - Len(Application.Substitute(strEmail, "@", ""))
'se houver mais de um @, o e-mail é inválido
If i <> 1 Then IsEmailValid = False: Exit Function
ReDim strArray(1 To 2)
'as duas próximas linhas colocam o texto à esquerda e à direita
'do símbolo @ em variáveis próprias
strArray(1) = Left(strEmail, InStr(1, strEmail, "@", 1) - 1)
strArray(2) = Application.Substitute(Right(strEmail, Len(strEmail) - _
    Len(strArray(1))), "@", "")

For Each strItem In strArray
    'verifica se há algo na variável.
'se não houver, falta então parte do e-mail
    If Len(strItem) <= 0 Then
        blnIsItValid = False
        IsEmailValid = blnIsItValid
        Exit Function
    End If
    'verifica apenas caracteres válidos no e-mail
    For i = 1 To Len(strItem)
'coloca todas as letras em caixa baixa (minúsculas) para uma verificação mais fácil
        c = LCase(Mid(strItem, i, 1))
        If InStr("abcdefghijklmnopqrstuvwxyz_-.", c) <= 0 _
        And Not IsNumeric(c) Then
            blnIsItValid = False
            IsEmailValid = blnIsItValid
            Exit Function
        End If
    Next i
'verifica se o primeiro caractere à esquerda e à direita não são pontos
    If Left(strItem, 1) = "." Or Right(strItem, 1) = "." Then
        blnIsItValid = False
        IsEmailValid = blnIsItValid
        Exit Function
    End If
```

```
Next strItem
'verifica se há um ponto na metade direita do endereço
If InStr(strArray(2), ".") <= 0 Then
    blnIsItValid = False
    IsEmailValid = blnIsItValid
    Exit Function
End If
i = Len(strArray(2)) - InStrRev(strArray(2), ".") 'localiza o período
'verifica se o número de letras corresponde a uma extensão de domínio válida
If i <> 2 And i <> 3 And i <> 4 Then
    blnIsItValid = False
    IsEmailValid = blnIsItValid
    Exit Function
End If
'verifica se não há dois pontos juntos no e-mail
If InStr(strEmail, "..") > 0 Then
    blnIsItValid = False
    IsEmailValid = blnIsItValid
    Exit Function
End If
IsEmailValid = blnIsItValid
End Function
```

Some as células com base na cor interna

Você tem uma lista de clientes e quanto cada um lhe deve. Você coloriu os valores dos clientes com pagamento em atraso há 30 dias e quer somar somente essas células.

> **NOTA** As células coloridas pela formatação condicional não funcionarão; elas devem ter uma cor interna.

```
SumColor(CellColor, SumRange)
```

Os argumentos são estes:

`CellColor` — O endereço de uma célula com a cor alvo

`SumRange` — O intervalo de células a ser pesquisado

```
Function SumByColor(CellColor As Range, SumRange As Range)
Dim myCell As Range
Dim iCol As Integer
Dim myTotal
iCol = CellColor.Interior.ColorIndex 'obtém a cor de destino
For Each myCell In SumRange 'verifica cada célula no intervalo designado
'se a cor da célula corresponder à cor alvo
If myCell.Interior.ColorIndex = iCol Then
'adiciona o valor da célula ao total
myTotal = WorksheetFunction.Sum(myCell) + myTotal
    End If
Next myCell
SumByColor = myTotal
End Function
```

A Figura 4.7 mostra uma planilha de exemplo que utiliza essa função.

Figura 4.7
Some células com base na cor interna.

Conte os valores únicos

Quantas vezes você tinha uma longa lista de valores e precisava conhecer quantos eram valores únicos? Essa função analisa um intervalo e informa exatamente isso, como mostrado na Figura 4.8:

```
NumUniqueValues(Rng)
```

Figura 4.8
Conte o número de valores únicos em um intervalo.

	A	B	C	D	E
1	A	11	=NumUniqueValues(A1:A16)		
2	R				
3	T				
4	A				
5	V				
6	F				
7	EE				
8	1				
9	19				
10	V				
11	Q				
12	V				
13	GE				
14	V				
15	123				
16	1				

O argumento é `Rng`, o intervalo para pesquisar valores únicos.

Exemplo de função:

```
Function NumUniqueValues(Rng As Range) As Long
Dim myCell As Range
Dim UniqueVals As New Collection
Application.Volatile 'força a função a recalcular quando o intervalo é modificado
On Error Resume Next
'o código a seguir coloca cada valor a partir do intervalo em uma coleção
'porque uma coleção, com um parâmetro-chave, pode conter apenas valores únicos,
'não haverá nenhuma duplicata, as instruções de erro forçam o programa a
'continuar quando as mensagens de erro aparecem para itens duplicados na coleção
For Each myCell In Rng
    UniqueVals.Add myCell.Value, CStr(myCell.Value)
Next myCell
On Error GoTo 0
'retorna o número de itens na coleção
NumUniqueValues = UniqueVals.Count
End Function
```

Remova duplicatas de um intervalo

Com que freqüência você precisou identificar em uma lista de itens apenas os valores únicos? A seguinte função analisa um intervalo e armazena apenas os valores únicos:

```
UniqueValues (OrigArray)
```

O argumento é `OrigArray`, um array a partir do qual remover duplicatas.

Essa primeira seção (declarações `Const`) deve estar na parte superior do módulo:

```
Const ERR_BAD_PARAMETER = "Parâmetro de array obrigatório"
Const ERR_BAD_TYPE = "Tipo Inválido"
Const ERR_BP_NUMBER = 20000
Const ERR_BT_NUMBER = 20001
```

Você pode colocar a seção de código a seguir em qualquer lugar do módulo, contanto que ela esteja abaixo da seção anterior:

```
Public Function UniqueValues(ByVal OrigArray As Variant) As Variant
    Dim vAns() As Variant
    Dim lStartPoint As Long
    Dim lEndPoint As Long
    Dim lCtr As Long, lCount As Long
    Dim iCtr As Integer
    Dim col As New Collection
    Dim sIndex As String
    Dim vTest As Variant, vItem As Variant
    Dim iBadVarTypes(4) As Integer
    'a função não funciona se o elemento do array for um dos
    'tipos a seguir
    iBadVarTypes(0) = vbObject
```

```
    iBadVarTypes(1) = vbError
    iBadVarTypes(2) = vbDataObject
    iBadVarTypes(3) = vbUserDefinedType
    iBadVarTypes(4) = vbArray
    'verifica se o parâmetro é um array
    If Not IsArray(OrigArray) Then
        Err.Raise ERR_BP_NUMBER, , ERR_BAD_PARAMETER
        Exit Function
    End If
    lStartPoint = LBound(OrigArray)
    lEndPoint = UBound(OrigArray)
    For lCtr = lStartPoint To lEndPoint
        vItem = OrigArray(lCtr)
        'primeiro verifica se o tipo de variável é aceitável
        For iCtr = 0 To UBound(iBadVarTypes)
            If VarType(vItem) = iBadVarTypes(iCtr) Or _
              VarType(vItem) = iBadVarTypes(iCtr) + vbVariant Then
                Err.Raise ERR_BT_NUMBER, , ERR_BAD_TYPE
                Exit Function
           End If
        Next iCtr
        'adiciona o elemento a uma coleção, utilizando-a como o índice
        'se ocorrer um erro, o elemento já existe
        sIndex = CStr(vItem)
        'primeiro elemento, adiciona automaticamente
        If lCtr = lStartPoint Then
            col.Add vItem, sIndex
            ReDim vAns(lStartPoint To lStartPoint) As Variant
            vAns(lStartPoint) = vItem
        Else
            On Error Resume Next
            col.Add vItem, sIndex
            If Err.Number = 0 Then
                lCount = UBound(vAns) + 1
                ReDim Preserve vAns(lStartPoint To lCount)
                vAns(lCount) = vItem
            End If
        End If
        Err.Clear
    Next lCtr
    UniqueValues = vAns
End Function
```

Eis um exemplo de como utilizar essa função. Veja Figura 4.9 para obter o resultado em uma planilha:

```
Function nodupsArray(rng As Range) As Variant
    Dim arr1() As Variant
    If rng.Columns.Count > 1 Then Exit Function
    arr1 = Application.Transpose(rng)
    arr1 = UniqueValues(arr1)
    nodupsArray = Application.Transpose(arr1)
End Function
```

C2 {=NoDupsArray($A$2:$A$17)}

	A	B	C	D	E	F
1	Origem		FunçãoMatriz			
2	A		A			
3	R		R			
4	T		T			
5	A		V			
6	V		F			
7	F		EE			
8	EE		1			
9	1		19			
10	19		Q			
11	V		GE			
12	Q		123			
13	V					
14	GE					
15	V					
16	123					
17	1					

Figura 4.9 Liste valores únicos a partir de um intervalo.

Localize a primeira célula de *comprimento* não-zero em um intervalo

Você importa uma grande lista de dados com muitas células vazias. Eis uma função que avalia um intervalo de células e retorna o valor da primeira célula de comprimento não-zero:

```
FirstNonZeroLength(Rng)
```

O argumento é `Rng`, o intervalo a ser pesquisado.

Exemplo de função:

```
Function FirstNonZeroLength(Rng As Range)
Dim myCell As Range
FirstNonZeroLength = 0#
For Each myCell In Rng
    If Not IsNull(myCell) And myCell <> "" Then
        FirstNonZeroLength = myCell.Value
        Exit Function
    End If
Next myCell
FirstNonZeroLength = myCell.Value
End Function
```

A Figura 4.10 mostra a função em uma planilha de exemplo.

Figura 4.10
Localize o valor da primeira célula de comprimento não-zero em um intervalo.

K22

	A	B	C	D
1		2		
2		=FirstNonZeroLengh(A1:A7)		
3	2			
4				
5	7			
6	8			
7	9			
8				

Substitua vários caracteres

O Excel tem uma função de substituição, mas é uma substituição de valor por valor. E se você tiver vários caracteres que precisem ser substituídos? A Figura 4.11 mostra vários exemplos de como essa função funciona:

```
MSubstitute(trStr, frStr, toStr)
```

Figura 4.11
Substitua vários caracteres em uma célula.

	A	B	C
1	1 Introdução	Introdução	=msubstitute(A1;"1";"")
2	Isso eran um teste	Isso era um teste	=msubstitute(A2;"eran"; "era")
3	123abc456	abc	=msubstitute(A3;"1234567890";"")
4	Oudytro Tieste	Outro Teste	=msubstitute(A4;"dyi";"")
5			

Os argumentos são estes:

`trStr` — A string a ser pesquisada
`frStr` — O texto sendo pesquisado
`toStr` — O texto de substituição

ATENÇÃO

Supõem-se que `toStr` tenha o mesmo comprimento que `frStr`. Se não tiver, os caracteres restantes são considerados nulos (`""`). A função diferencia letras maiúsculas de minúsculas. Para substituir todas as instâncias de a, utilize a e A. Não é possível substituir um caractere por dois caracteres. Este

```
=MSUBSTITUTE("This is a test","i","$@")
```

resulta nisto:

```
"Th$s $s a test"
```

Exemplo de função:

```
Function MSUBSTITUTE(ByVal trStr As Variant, frStr As String, _
        toStr As String) As Variant
Dim iCol As Integer
Dim j As Integer
Dim Ar As Variant
Dim vfr() As String
Dim vto() As String
ReDim vfr(1 To Len(frStr))
ReDim vto(1 To Len(frStr))
'coloca as strings em um array
For j = 1 To Len(frStr)
    vfr(j) = Mid(frStr, j, 1)
    If Mid(toStr, j, 1) <> "" Then
        vto(j) = Mid(toStr, j, 1)
    Else
        vto(j) = ""
    End If
Next j
'compara cada caractere e substitui se necessário
If IsArray(trStr) Then
    Ar = trStr
    For iRow = LBound(Ar, 1) To UBound(Ar, 1)
        For iCol = LBound(Ar, 2) To UBound(Ar, 2)
            For j = 1 To Len(frStr)
                Ar(iRow, iCol) = Application.Substitute(Ar(iRow, iCol), _
            vfr(j), vto(j))
            Next j
        Next iCol
    Next iRow
Else
    Ar = trStr
    For j = 1 To Len(frStr)
        Ar = Application.Substitute(Ar, vfr(j), vto(j))
    Next j
End If
MSUBSTITUTE = Ar
End Function
```

Recupere números a partir de texto misto

Essa função extrai e retorna números a partir do texto que é uma mistura de números e letras, conforme mostrado na Figura 4.12:

```
RetrieveNumbers (myString)
```

Figura 4.12
Extrai números do texto misto.

G10 =RetrieveNumbers(A1)

	A	B	C	D	E
1	123abc456	123456			
2	1b2k3j34ioj	12334			
3	123abc456	123			
4					

O argumento é `myString`, o texto que contém os números a ser extraídos.

Exemplo de função:

```
Function RetrieveNumbers(myString As String)
Dim i As Integer, j As Integer
Dim OnlyNums As String
'iniciando no FIM da string e movendo-se para trás (Step -1)
For i = Len(myString) To 1 Step -1
'IsNumeric é uma função VBA que retorna True se uma variável for um número
'quando um número é localizado, ele é adicionado à string OnlyNums
    If IsNumeric(Mid(myString, i, 1)) Then
        j = j + 1
        OnlyNums = Mid(myString, i, 1) & OnlyNums
    End If
    If j = 1 Then OnlyNums = CInt(Mid(OnlyNums, 1, 1))
Next i
RetrieveNumbers = CLng(OnlyNums)
End Function
```

Converta o número da semana *em data*

Você já recebeu um relatório de planilha em que todos os cabeçalhos exibiam o número da semana? Não sei se você sabe, mas eu realmente não sei o que significa 'Week 15'. Eu teria de utilizar outro calendário e contar as semanas. E se for preciso examinar um ano anterior? O que precisamos é de uma pequena função que converta 'Week ## Year' na data da segunda-feira dessa semana, conforme mostrado na Figura 4.13:

```
Weekday(Str)
```

Figura 4.13
Converta um número da semana em uma data mais fácil de referenciar.

	A	B	C
1	Week 15 2007	9 de abril de 2007	
2		=convertWEEKDAY(A1)	
3			

O argumento é `Str`, a semana a ser convertida no formato `"Week ##, YYYY"`.

NOTA O resultado precisa ser formatado como uma data.

Exemplo de função:

```
Function ConvertWeekDay(Str As String) As Date
Dim Week As Long
Dim FirstMon As Date
Dim TStr As String
FirstMon = DateSerial(Right(Str, 4), 1, 1)
FirstMon = FirstMon - FirstMon Mod 7 + 2
TStr = Right(Str, Len(Str) - 5)
Week = Left(TStr, InStr(1, TStr, " ", 1)) + 0
ConvertWeekDay = FirstMon + (Week - 1) * 7
End Function
```

Separe string delimitada

Nesse exemplo, você precisa colar uma coluna de dados delimitados. Você poderia utilizar o Texto para Colunas do Excel, mas este analisa sintaticamente a coisa toda, e você só precisa de um ou dois elementos a partir de cada célula.

O que você precisa é de uma função que permita especificar o número do elemento na string de que precisa, como mostrado na Figura 4.14:

```
StringElement(str,chr,ind)
```

Figura 4.14
Extraindo um elemento único do texto delimitado.

	A	B	C	D	E	F	G	H
1					ind			
2	str	char	1	2	3	4	5	6
3	A|B|C|D|E|F	|	A	B	C	D	E	F
4			=StringElement($A$3,$B$3,C2)					
5								

Os argumentos são estes:

`str` — A string a ser analisada sintaticamente

`chr` — O delimitador

`ind` — A posição do elemento a ser retornado

Exemplo de função:

```
Function StringElement(str As String, chr As String, ind As Integer)
Dim arr_str As Variant
arr_str = Split(str, chr) 'Não é compatível com XL97
StringElement = arr_str(ind - 1)
End Function
```

Classifique e concatene

O que dizer de pegar uma coluna de dados, classificá-la e concatená-la, usando uma vírgula (,) como delimitador (veja Figura 4.15)?

```
SortConcat(Rng)
```

Figura 4.15
Classifique e concatene um intervalo de variáveis.

	A	B	C
1	Lista não Ordenada	String Ordenada	
2	q	1,14,50,9,f,gg,q,r,rrrrr	
3	r	=sortConcat(A2:A11)	
4	f		
5	a		
6	gg		
7	1		
8	9		
9	50		
10	14		
11	rrrrr		
12			

O argumento é `Rng`, o intervalo de dados a ser classificado e concatenado. `SortConcat` chama outro procedimento, `BubbleSort`, que deve ser incluído.

Exemplo de função:

```
Function SortConcat(Rng As Range) As Variant
Dim MySum As String, arr1() As String
Dim j As Integer, i As Integer
Dim cl As Range
Dim concat As Variant
On Error GoTo FuncFail:
'inicializa a saída
SortConcat = 0#
'evita questões de usuário
If Rng.Count = 0 Then Exit Function
'posiciona o intervalo na variável variante que contém o array
ReDim arr1(1 To Rng.Count)
'preenche o array
i = 1
For Each cl In Rng
    arr1(i) = cl.Value
    i = i + 1
Next
'classifica elementos do array
Call BubbleSort(arr1)
'cria a string a partir dos elementos de array
For j = UBound(arr1) To 1 Step -1
    If Not IsEmpty(arr1(j)) Then
        MySum = arr1(j) & ", " & MySum
    End If
Next j
'atribui valor à função
SortConcat = Left(MySum, Len(MySum) - 2)
'ponto de saída
concat_exit:
Exit Function
'exibe o erro na célula
FuncFail:
SortConcat = Err.Number & " - " & Err.Description
Resume concat_exit
End Function
```

A função a seguir é a popular `BubbleSort`, um programa utilizado por muitos para fazer uma classificação simples dos dados:

```
Sub BubbleSort(List() As String)
'  classifica o array List em ordem crescente
Dim First As Integer, Last As Integer
Dim i As Integer, j As Integer
Dim Temp
First = LBound(List)
Last = UBound(List)
For i = First To Last - 1
    For j = i + 1 To Last
        If UCase(List(i)) > UCase(List(j)) Then
            Temp = List(j)
            List(j) = List(i)
```

```
            List(i) = Temp
        End If
    Next j
Next i
End Sub
```

Classifique caracteres numéricos e alfabéticos

Essa função seleciona um intervalo misto de caracteres numéricos e alfabéticos e os classifica — primeiro numericamente e então alfabeticamente. O resultado é colocado em um array que pode ser exibido em uma planilha por meio do uso de uma fórmula de array, como mostrado na Figura 4.16:

```
sorter(Rng)
```

Figura 4.16
Classifique uma lista mista alfanumérica.

B2 {=Sorter($A$2:$A$14)}

	A	B	C
1	dados iniciais	dados classificados	
2	E	2	
3	B	3	
4	Y	6	
5	T	9	
6	R	9d	
7	F	B	
8	SS	DD	
9	DD	E	
10	9	F	
11	3	R	
12	2	SS	
13	6	T	
14	9d	Y	
15			

O argumento é `Rng`, o intervalo a ser classificado.
Exemplo de função:

```
Function sorter(Rng As Range) As Variant
'retorna um array
Dim arr1() As Variant
If Rng.Columns.Count > 1 Then Exit Function
arr1 = Application.Transpose(Rng)
QuickSort arr1
sorter = Application.Transpose(arr1)
End Function
```

A função usa os dois procedimentos a seguir para classificar os dados no intervalo:

```
Public Sub QuickSort(ByRef vntArr As Variant,
    Optional ByVal lngLeft As Long = -2, _
    Optional ByVal lngRight As Long = -2)
Dim i, j, lngMid As Long
Dim vntTestVal As Variant
If lngLeft = -2 Then lngLeft = LBound(vntArr)
If lngRight = -2 Then lngRight = UBound(vntArr)
If lngLeft < lngRight Then
    lngMid = (lngLeft + lngRight) \ 2
    vntTestVal = vntArr(lngMid)
    i = lngLeft
    j = lngRight
    Do
        Do While vntArr(i) < vntTestVal
            i = i + 1
        Loop
        Do While vntArr(j) > vntTestVal
            j = j - 1
        Loop
        If i <= j Then
            Call SwapElements(vntArr, i, j)
            i = i + 1
            j = j - 1
        End If
    Loop Until i > j
    If j <= lngMid Then
        Call QuickSort(vntArr, lngLeft, j)
        Call QuickSort(vntArr, i, lngRight)
    Else
        Call QuickSort(vntArr, i, lngRight)
```

```
        Call QuickSort(vntArr, lngLeft, j)
    End If
End If
End Sub

Private Sub SwapElements(ByRef vntItems As Variant,
    ByVal lngItem1 As Long, _
    ByVal lngItem2 As Long)
Dim vntTemp As Variant
vntTemp = vntItems(lngItem2)
vntItems(lngItem2) = vntItems(lngItem1)
vntItems(lngItem1) = vntTemp
End Sub
```

Procure uma string no texto

Você já precisou descobrir quais células contêm uma string de texto específica? A função a seguir pode pesquisar strings em um intervalo, à procura do texto especificado. Ela retorna um resultado que identifica quais células contêm o texto, conforme mostrado na Figura 4.17:

```
ContainsText(Rng,Text)
```

Figura 4.17
Retorne um resultado que identifica que célula(s) contém(êm) uma string especificada.

	A	B	C	D	E
1	Isto é uma maçã	A3	=ConatinsText(A1:A3,"banana")		
2	Isto é uma laranja	A1,A2	=ConatinsText(A1:A3,"Isto é")		
3	Isto é uma banana				
4					

Os argumentos são estes:

`Rng` — O intervalo em que pesquisar

`Text` — O texto em que pesquisar

Exemplo de função:

```
Function ContainsText(Rng As Range, Text As String) As String
Dim T As String
Dim myCell As Range
For Each myCell In Rng 'verifica cada célula
    If InStr(myCell.Text, Text) > 0 Then 'verifica se o texto está na string
        If Len(T) = 0 Then 'se o texto for encontrado, inclui o endereço no meu resultado
            T = myCell.Address(False, False)
        Else
            T = T & "," & myCell.Address(False, False)
        End If
    End If
Next myCell
ContainsText = T
End Function
```

Inverta o conteúdo de uma célula

Essa função é bastante divertida, mas talvez você a considere útil — ela inverte o conteúdo de uma célula:

```
ReverseContents(myCell, IsText)
```

Os argumentos são estes:

`myCell` — A célula especificada

`IsText` — (Opcional) Se o valor da célula deve ser tratado como texto (padrão) ou um número

Exemplo de função:

```
Function ReverseContents(myCell As Range, Optional IsText As Boolean = True)
Dim i As Integer
```

```
Dim OrigString As String, NewString As String
OrigString = Trim(myCell) 'remove espaços no início e no final
For i = 1 To Len(OrigString)
'adicionando a variável NewString ao caractere,
'em vez de adicionar o caractere à NewString, a string é invertida
    NewString = Mid(OrigString, i, 1) & NewString
Next i
If IsText = False Then
    ReverseContents = CLng(NewString)
Else
    ReverseContents = NewString
End If
End Function
```

Retorne múltiplos valores máximos

MAX localiza e retorna o valor máximo em um intervalo, mas não informa se há mais de um valor máximo. Essa função retorna o(s) endereço(s) do valor(es) máximo(s) em um intervalo, como mostrado na Figura 4.18:

```
ReturnMaxs(Rng)
```

O argumento é Rng, o intervalo no qual procurar o(s) valor(es) máximo(s).

Figura 4.18
Retorne os endereços de todos os valores máximos em um intervalo.

	A	B	C
1	3	A9,A11	
2	9	=ReturnMaxs(A1:A15)	
3	5		
4	6		
5	6		
6	7		
7	6		
8	3		
9	10		
10	4		
11	10		
12	6		
13	9		
14	7		
15	1		
16			

Exemplo de função:

```
Function ReturnMaxs(Rng As Range) As String
Dim Mx As Double
Dim myCell As Range
'se houver apenas uma célula no intervalo, então feche
If Rng.Count = 1 Then ReturnMaxs = Rng.Address(False, False): Exit Function
Mx = Application.Max(Rng) 'usa o Max do Excel para localizer o valor máximo no intervalo
'como você sabe agora qual é o valor max
'pesquisa o intervalo procurando correspondências e retorna o endereço
For Each myCell In Rng
    If myCell = Mx Then
        If Len(ReturnMaxs) = 0 Then
            ReturnMaxs = myCell.Address(False, False)
        Else
            ReturnMaxs = ReturnMaxs & ", " & myCell.Address(False, False)
        End If
    End If
Next myCell
End Function
```

Retorne o endereço de hiperlink

Você recebeu uma planilha com uma lista de informações com hiperlinks e quer ver os links reais, não o texto descritivo. Você poderia simplesmente clicar com o botão direito do mouse neles e escolher Editar Hiperlink, mas você quer algo mais permanente. Essa função extrai o endereço de hiperlink, como mostrado na Figura 4.19:

```
GetAddress(Hyperlink)
```

Figura 4.19
Extraia o endereço que está por trás de um hiperlink.

	A	B	C	D
1	Tracy Syrstad	Tracy@MrExcel.com	=GetAddress(A1)	
2	O melhor site sobre Excel	http://mrexcel.com/		
3				

O argumento é `Hyperlink`, a célula com hiperlink a partir do qual você quer extrair o endereço.
Exemplo de função:

```
Function GetAddress(HyperlinkCell As Range)
    GetAddress = Replace(HyperlinkCell.Hyperlinks(1).Address, "mailto:", "")
End Function
```

Retorne a letra da coluna de um endereço de célula

Você pode usar `CELL("Col")` para obter um número de coluna; mas, e se você precisar da letra da coluna? Essa função extrai a letra de coluna a partir de um endereço de célula, como mostrado na Figura 4.20:

```
ColName(Rng)
```

Figura 4.20
Retorne a letra de coluna a partir de um endereço de célula.

	A	B
1	A	=ColName(A1)
2	XFD	=ColName(XFD1048576)
3		

O argumento é `Rng`, a célula para a qual você quer a letra de coluna.
Exemplo de função:

```
Function ColName(Rng As Range) As String
ColName = Left(Rng.Range("A1").Address(True, False), _
    InStr(1, Rng.Range("A1").Address(True, False), "$", 1) - 1)
End Function
```

Aleatório estático

A função `=RAND()` pode ser muito útil para criar números aleatórios, mas ela recalcula constantemente. E se você precisar de números aleatórios, mas não quiser que eles mudem constantemente? A função a seguir posiciona um número aleatório, mas ele só muda se você forçar a célula a recalcular, como mostrado na Figura 4.21:

```
StaticRAND()
```

Figura 4.21
Produza números aleatórios que não são tão voláteis.

	A	B
1	0.129781902	=StaticRand()
2	4.121.850.133,00	=StaticRand()-100
3	34.82	=StaticRand()-100
4		

Não há nenhum argumento para essa função.
Exemplo de função:

```
Function StaticRAND() As Double
Randomize
STATICRAND = Rnd
End Function
```

Usando `Select Case` em uma planilha

Você alguma vez já aninhou uma instrução `If...Then...Else` em uma planilha para retornar um valor? A instrução `Select...Case` disponível no VBA torna isso muito mais fácil, mas você não pode utilizar instruções `Select...Case` em uma fórmula de planilha. Em vez disso, é possível criar uma FDU (veja Figura 4.22).

	A	B	C	D
1	Mês	Ano		
2	4	2007		
3				
4	período:			
5	1º de Outubro 2006 a 31 de Outubro 2006			
6	=State_Period(A2;B2)			
7				

Figura 4.22
Exemplo de como usar uma estrutura `Select...Case` em uma FDU em vez de instruções `If...Then` aninhadas.

A função a seguir mostra como utilizar instruções `Select` para produzir os resultados de uma instrução `If...Then` aninhada:

```
Function state_period(mth As Integer, yr As Integer)
Select Case mth
  Case 1
     state_period = "1º de Julho, " & yr - 1 & " a 31 de Julho, " & yr - 1
  Case 2
     state_period = "1º de Agosto, " & yr - 1 & " a 31 de Agosto, " & yr - 1
  Case 3
     state_period = "1º de Setembro, " & yr - 1 & " a 30 de Setembro, " & yr - 1
  Case 4
     state_period = " 1º de Outubro, " & yr - 1 & " a 31 de Outubro, " & yr - 1
  Case 5
     state_period = " 1º de Novembro, " & yr - 1 & " a 30 de Novembro, " & yr - 1
  Case 6
     state_period = " 1º de Dezembro, " & yr - 1 & " a 31 de Dezembro, " & yr - 1
  Case 7
     state_period = " 1º de Janeiro, " & yr & " a 31 de Janeiro, " & yr
  Case 8
     state_period = " 1º de Fevereiro, " & yr & " a 28 de Fevereiro, " & yr
  Case 9
    state_period = " 1º de Março, " & yr & " a 31 de Março, " & yr
  Case 10
    state_period = " 1º de Abril, " & yr & " a 30 de Abril, " & yr
  Case 11
    state_period = " 1º de Maio, " & yr & " a 31 de Maio, " & yr
  Case 12
    state_period = " 1º de Junho, " & yr & " a 30 de Junho, " & yr
  Case 13
    state_period = "Pre-Final"
  Case 14
    state_period = "Closeout"
End Select
End Function
```

Próximos passos

O próximo capítulo descreve um componente fundamental de qualquer linguagem de programação: loops. Se você já participou de alguma aula de programação, deve conhecer estruturas básicas de loops, então verá que o VBA suporta todos os loops comuns. Você também verá um loop especial, `For Each...Next`, que é o único destinado à programação orientada a objetos, como o VBA.

Loops e controle de fluxo

5

NESTE CAPÍTULO

Loops são um componente fundamental de qualquer linguagem de programação. Se você fez algum curso de programação, até mesmo de BASIC, provavelmente encontrou um loop `For...Next`. Felizmente, o VBA suporta todos os loops normais, além de um loop especial, cujo uso é excelente com o VBA.

Este capítulo abrange as construções básicas de loop:

- `For...Next`
- `Do...While`
- `Do...Until`
- `While...Loop`
- `Until...Loop`

Também discutimos a excelente construção de um loop exclusivo para linguagens orientadas a objetos:

- `For Each...Next`

Loops `For...Next`

`For` e `Next` são construções de loop comuns. Tudo entre `For` e `Next` é executado várias vezes. Toda vez que o código executar, uma determinada variável de contador (especificada na instrução `For`) terá um valor diferente.

Considere este código:

```
For I = 1 to 10
    Cells(I, I).Value = I
Next I
```

Quando esse programa começar a executar, atribuímos à variável um nome I. Na primeira passagem pelo código, a variável I é definida como 1. Na primeira vez que o loop é executado, ele é igual a 1, assim a célula na Linha 1, Coluna 1 será configurada como 1 (veja Figura 5.1).

Figura 5.1
Depois da primeira iteração pelo loop, a célula na Linha 1, Coluna 1 tem o valor de 1.

	A	B	C	D
1	1			
2				
3				
4				

Vamos examinar mais detalhadamente o que acontece quando o VBA encontra a linha que diz `Next I`. Antes de executar essa linha, a variável `I` é igual a 1. Durante a execução de `Next I`, o VBA deve tomar uma decisão: adicionar 1 à variável `I` e compará-la com o valor máximo na cláusula `To` da instrução `For`. Se estiver dentro dos limites especificados na cláusula `To`, o loop não é terminado. Nesse caso, o valor de `I` será incrementado para 2. A execução do código volta então à primeira linha do código depois da instrução `For`. A Figura 5.2 mostra o estado do programa antes de executar a linha `Next`. A Figura 5.3 mostra o que acontece depois dessa execução.

Figura 5.2
Antes de executar a instrução `Next I`, `I` é igual a 1. O VBA pode adicionar com segurança 1 à `I` e ela será menor do que o 10 especificado na cláusula `To` da instrução `For`.

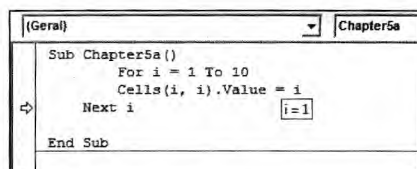



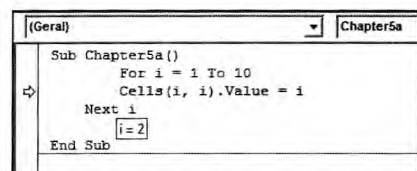
Figura 5.3
Depois de executar a instrução `Next I`, `I` é incrementado para 2. A execução do código continua na linha de código após a instrução `For`.

Na segunda passagem pelo loop, o valor de `I` é 2. A célula na Linha 2, Coluna 2 (isto é, a célula B2) obtém o valor 2 (veja Figura 5.4).

Figura 5.4
Depois da segunda passagem pelo loop, as células A1 e B2 são preenchidas.

À medida que o processo continua, a instrução `Next I` avança `I` até 3, depois até 4 e assim por diante. Na décima passagem pelo loop, o valor 10 é atribuído à célula na Linha 10, Coluna 10.

É interessante ver o que acontece com a variável `I` na última passagem por `Next I`. Na Figura 5.5, você pode ver que, antes de executar `Next I` pela décima vez, a variável `I` é igual a 10.

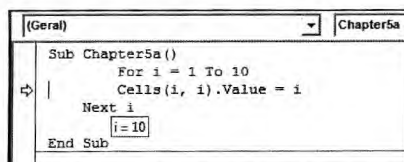
Figura 5.5
Antes de executar `Next I` pela décima vez, a variável `I` é igual a 10.

O VBA agora está em um ponto de decisão. Ele adiciona 1 à variável `I`. Agora, `I` é igual a 11, que é maior do que o limite no loop `For...Next`. O VBA então passa a execução para a próxima linha na macro depois da instrução `Next` (veja Figura 5.6). Caso você esteja tentado a utilizar a variável `I` mais tarde na macro, é importante perceber que ela poderia ser incrementada para além do limite especificado na cláusula `To` da instrução `For`.

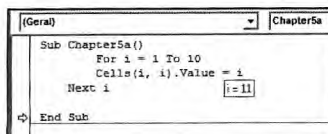
Figura 5.6
Depois de incrementar `I` para 11, a execução do código passa para a linha depois da instrução `Next`.

No final do loop, você obtém o resultado mostrado na Figura 5.7.

Figura 5.7
Depois que o loop completa as dez iterações, esse é o resultado.

	A	B	C	D	E	F	G	H	I	J	K
1	1										
2		2									
3			3								
4				4							
5					5						
6						6					
7							7				
8								8			
9									9		
10										10	
11											

O uso comum para esse loop é investigar todas as linhas em um conjunto de dados e decidir pela realização de alguma ação com base em determinados critérios. Por exemplo, se quisesse marcar todas as linhas com receitas positivas de serviço na coluna F, você poderia usar este loop:

```
For I = 2 to 10
    If Cells(I, 6).Value > 0 Then
        Cells(I, 8).Value = "ReceitaServiço"
        Cells(I, 1).Resize(1, 8).Interior.ColorIndex = 4
    End If
Next i
```

Esse loop verifica todos os itens de dados da Linha 2 à Linha 10. Se houver um número positivo na coluna F, a coluna H dessa linha terá um novo rótulo e as células nas colunas A:H da linha serão pintadas de verde. Depois de executar essa macro, os resultados se parecerão com a Figura 5.8.

Figura 5.8
Depois que o loop completar nove iterações, todas as linhas com valores positivos na coluna F são pintados de verde e o rótulo "Receita Servico" será adicionado à Coluna H.

A1 fx DataFatura

	A	B	C	D	E	F	G	H	I
1	DataFatura	NumeroFatura	NumeroVendedor	NumeroCliente	ReceitaProduto	ReceitaServico	CustoProduto		
2	09/06/2008	123829	S21	C8754	21000	0	9875		
3	09/06/2008	123834	S54	C7796	339000	0	195298		
4	09/06/2008	123835	S21	C1654	161000	0	90761		
5	09/06/2008	123836	S45	C6460	275500	10000	146341	Receita Servico	
6	09/06/2008	123837	S54	C5143	925400	0	473515		
7	09/06/2008	123841	S21	C8361	94400	0	53180		
8	09/06/2008	123842	S45	C1842	36500	55000	20696	Receita Servico	
9	09/06/2008	123843	S54	C4107	599700	0	276718		
10	09/06/2008	123844	S21	C5205	244900	0	143393		

Utilizando variáveis na instrução `For`

O exemplo anterior não é muito útil porque só funciona quando há exatamente dez linhas de dados. É possível utilizar uma variável para especificar o limite superior/inferior da instrução `For`. Esse exemplo de código identifica `FinalRow` com dados e então faz um loop a partir da Linha 2 até essa linha:

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
For I = 2 to FinalRow
    If Cells(I, 6).Value > 0 Then
        Cells(I, 8).Value = "ReceitaServiço"
        Cells(I, 1).Resize(1, 8).Interior.ColorIndex = 4
    End If
Next I
```

Tenha cuidado ao utilizar variáveis. E se o arquivo importado hoje estiver vazio e tiver apenas uma linha de título? Nesse caso, a variável `FinalRow` será igual a 1. Isso torna a primeira instrução do loop essencialmente, digamos, 'For I = 2 a 1'. Como o número inicial é mais alto do que o final, o loop não executa de jeito nenhum. A variável I é igual a 2 e a execução do código pulará para a linha depois de Next.

Variações no loop `For...Next`

Em um loop `For...Next` é possível fazer a variável de loop pular até alguma coisa além de 1. Você poderia utilizá-la para aplicar uma formatação de barra verde a uma série de linhas alternadas em um conjunto de dados, por exemplo. Nesse caso, seria recomendável que a variável de contador `I` examinasse linhas alternadas no conjunto de dados. Indique isso adicionando a cláusula `Step` ao final da instrução `For`:

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
For i = 2 to FinalRow Step 2
    Cells(i, 1).Resize(1, 8).Interior.ColorIndex = 35
Next i
```

Ao executar esse código, o VBA adiciona um sombreado verde-claro às Linhas 2, 4, 6 e assim por diante (veja Figura 5.9).

Figura 5.9
A cláusula `Step` na instrução `For` do loop faz com que a ação ocorra em linhas alternadas.

	A	B	C	D	E	F	G	H
1	DataFatura	NumeroFatura	NumeroVendedor	NumeroCliente	ReceitaProduto	ReceitaServico	CustoProduto	
2	07/06/2004	123829	S21	C8754	21000	0	9875	
3	07/06/2004	123830	S45	C3390	188100	0	85083	
4	07/06/2004	123831	S54	C2523	510600	0	281158	
5	07/06/2004	123832	S21	C5519	86200	0	49967	
6	07/06/2004	123833	S45	C3245	800100	0	388277	
7	07/06/2004	123834	S54	C7796	339000	0	195298	
8	07/06/2004	123835	S21	C1654	161000	0	90761	
9	07/06/2004	123836	S45	C6460	275500	10000	146341	
10	07/06/2004	123837	S54	C5143	925400	0	473515	
11	07/06/2004	123838	S21	C7868	148200	0	75700	
12	07/06/2004	123839	S45	C3310	890200	0	468333	
13	07/06/2004	123840	S54	C2959	986000	0	528980	
14	07/06/2004	123841	S21	C8361	94400	0	53180	
15	07/06/2004	123842	S45	C1842	36500	55000	20696	
16	07/06/2004	123843	S54	C4107	599700	0	276718	
17	07/06/2004	123844	S21	C5205	244900	0	143393	
18	07/06/2004	123845	S45	C7745	63000	0	35102	
19	07/06/2004	123846	S54	C1730	212600	0	117787	
20	07/06/2004	123847	S21	C6292	974700	0	478731	
21	07/06/2004	123848	S45	C2008	327700	0	170968	
22	07/06/2004	123849	S54	C4096	30700	0	18056	
23								

A cláusula `Step` pode ser facilmente *um número* qualquer. Talvez você queira verificar cada décima linha de um conjunto de dados para extrair um exemplo aleatório. Nesse caso, utilize `Step 10`:

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
NextRow = FinalRow + 5
Cells(NextRow-1, 1).Value = "Random Sample of Above Data"
For I = 2 to FinalRow Step 10
    Cells(I, 1).Resize(1, 8).Copy Destination:=Cells(NextRow, 1)
    NextRow = NextRow + 1
Next i
```

Você também pode fazer um loop `For...Next` executar retroativamente de cima para baixo. Isso é particularmente útil se você excluir linhas seletivamente. Para fazer isso, inverta a ordem da instrução `For` e faça a cláusula `Step` especificar um número negativo:

```
'Exclua todas as linhas em que a coluna C é a representante interna - S54
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
For I = FinalRow to 2 Step -1
    If Cells(I, 3).Value = "S54" Then
        Cells(I, 1).EntireRow.Delete
    End If
Next i
```

Fechando um loop logo depois de uma condição ser atendida

Às vezes você não precisa executar o loop inteiro. Talvez só precise ler o conjunto de dados até localizar um registro que atenda certos critérios. Nesse caso, você quer encontrar o primeiro registro e então parar o loop. Uma instrução chamada `Exit For` faz isso.

A macro do exemplo a seguir procura uma linha no conjunto de dados, na qual a receita do serviço na coluna F seja positiva e a receita do produto na coluna E seja 0. Se essa linha for localizada, você poderá adicionar uma mensagem informando que o arquivo precisa de processamento manual hoje e mover o ponteiro de célula para esta linha:

```
'Há alguma situação de processamento especial nos dados?
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
ProblemFound = False
For I = 2 to FinalRow
    If Cells(I, 6).Value > 0 Then
        If cells(I, 5).Value = 0 Then
            Cells(I, 6).Select
            ProblemFound = True
            Exit For
        End If
    End If
Next I
If ProblemFound Then
    MsgBox "Há um problema na coluna " & I
    Exit Sub
End
```

Aninhando um loop dentro de outro loop

Não há problemas em executar um loop dentro de outro loop. Talvez o primeiro loop esteja executando por todas as linhas no seu recordset. O segundo loop poderia então executar por todas as colunas. Considere o conjunto de dados mostrado na Figura 5.10.

Figura 5.10
Aninhando um loop dentro do outro, o VBA pode fazer um loop por cada linha e então por cada coluna.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Item	Janeiro	Fevereiro	Março	Abril	Maio	Junho	Julho	Agosto	Setembro	Outubro	Novembro	Dezembro
2	Receita de Hardware	1.972.637	1.655.321	1.755.234	1.531.060	1.345.699	1.172.668	1.644.299	1.538.936	1.975.409	1.197.156	1.144.025	1.533.728
3	Receita de Software	236.716	198.639	210.628	183.727	161.484	140.720	197.316	184.672	237.049	143.659	137.283	184.047
4	Receita de Serviços	473.433	397.277	421.256	367.454	322.968	281.440	394.632	369.345	474.098	287.317	274.566	368.095
5	Custo de Mercadorias Vendidas	1.084.951	910.427	965.379	842.083	740.135	644.967	904.364	846.415	1.086.475	658.436	629.214	843.550
6	Despesas de Vendas	394.527	331.064	351.047	306.212	269.140	234.534	328.860	307.787	395.082	239.431	228.805	306.746
7	Custos Gerais e Administrativos	150.000	150.000	150.000	150.000	150.000	150.000	150.000	150.000	150.000	150.000	150.000	150.000
8	Pesquisa e Desenvolvimento	125.000	125.000	125.000	125.000	125.000	125.000	125.000	125.000	125.000	125.000	125.000	125.000

```
'Faz um loop por cada linha e coluna
'Adiciona um formato de tabuleiro de damas
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
FinalCol = Cells(1, 255).End(xlToLeft).Column
For I = 2 to FinalRow
    ' Para as linhas pares, inicia na coluna 1
    ' Para as linhas ímpares, inicia na coluna 2
    If I Mod 2 = 1 Then
        StartCol = 1
```

```
        Else
            StartCol = 2
        End If
        For J = StartCol to FinalCol Step 2
            Cells(I, J).Interior.ColorIndex = 35
        Next J
    Next I
```

Nesse código, o loop externo utiliza a variável de contador I para fazer um loop por todas as linhas no conjunto de dados. O loop interno utiliza a variável de contador J para fazer um loop por todas as colunas nessa linha. Como a Figura 5.10 tem sete linhas de dados, o código executa sete vezes pelo loop I. Todas as vezes pelo loop I, o código passa rapidamente seis ou sete vezes pelo loop J. Isso significa que a linha do código que está dentro do loop J acaba sendo executada várias vezes a cada passagem pelo loop I. A Figura 5.11 mostra o resultado.

Figura 5.11
O resultado de aninhar um loop dentro do outro; o VBA faz um loop por cada linha e então por cada coluna.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Item	Janeiro	Fevereiro	Março	Abril	Maio	Junho	Julho	Agosto	Setembro	Outubro	Novembro	Dezembro
2	Receita de Hardware	1.972.637	1.655.321	1.755.234	1.531.060	1.345.699	1.172.668	1.644.299	1.538.936	1.975.409	1.197.156	1.144.025	1.533.728
3	Receita de Software	236.716	198.639	210.628	183.727	161.484	140.720	197.316	184.672	237.049	143.659	137.283	184.047
4	Receita de Serviços	473.433	397.277	421.256	367.454	322.968	281.440	394.632	369.345	474.098	287.317	274.566	368.095
5	Custo de Mercadorias Vendidas	1.084.951	910.427	965.379	842.083	740.135	644.967	904.364	846.415	1.086.475	658.436	629.214	843.550
6	Despesas de Vendas	394.527	331.064	351.047	306.212	269.140	234.534	328.860	307.787	395.082	239.431	228.805	306.746
7	Custos Gerais e Administrativos	150.000	150.000	150.000	150.000	150.000	150.000	150.000	150.000	150.000	150.000	150.000	150.000
8	Pesquisa e Desenvolvimento	125.000	125.000	125.000	125.000	125.000	125.000	125.000	125.000	125.000	125.000	125.000	125.000

Loops Do

Há diversas variações do loop `Do`. O loop `Do` mais básico é excelente para fazer várias tarefas triviais. Uma vez alguém me enviou uma lista de endereços, um abaixo do outro em uma coluna, como mostrado na Figura 5.12.

Figura 5.12
Alguém enviou endereços nesse formato, mas seria mais útil colocá-los em um formato de banco de dados para que possam ser posteriormente usados em uma mala-direta.

	A
1	
2	John Smith
3	123 Main Street
4	Akron OH 44308
5	
6	Jane Doe
7	245 State Street
8	Chicago IL 60011
9	
10	Ralph Emerson
11	345 2nd Ave
12	New York NY 10011
13	
14	George Washington
15	456 3rd St
16	Philadelphia PA 12345
17	
18	John Adams
19	543 4th Ave
20	Baltimore MD 12345
21	
22	Thomas Jefferson
23	654 5th Ave
24	Roanoke VA 12345
25	
26	James Madison
27	987 6th St
28	Hollywood CA 12345
29	

Precisei reorganizar esses endereços em um banco de dados com os nomes na Coluna B, as ruas na Coluna C e as cidades e estados na Coluna D. Configurando Gravação Relativa (ver o Capítulo 1, "Libere o poder do Excel com o VBA") e utilizando a tecla de atalho Ctrl+A, gravei essa pequena parte do código útil. O código foi projetado para copiar um único endereço no formato de banco de dados. Ele também move o ponteiro da célula até o nome do próximo endereço na lista. Isso permitiu que eu permanecesse na minha mesa e, toda vez que pressionasse Ctrl+A, ele reformataria um endereço para mim.

➔ Para uma discussão sobre gravação relativa, **consulte** "Uma possível solução: usando referências relativas ao gravar," **p. 25**, no Capítulo 1.

```
Sub Macro3()
'
' Macro3 Macro
' macro gravada em 10/29/2003 por Bill Jelen
' move um endereço para o formato do banco de dados.
' move então o ponteiro de célula até o início do próximo endereço.
'
' Atalhos pelo teclado: Ctrl+Shift+A
```

```
    Selection.Copy
    ActiveCell.Offset(0, 1).Range("A1").Select
    ActiveSheet.Paste
    ActiveCell.Offset(1, -1).Range("A1").Select
    Application.CutCopyMode = False
    Selection.Copy
    ActiveCell.Offset(-1, 2).Range("A1").Select
    ActiveSheet.Paste
    ActiveCell.Offset(2, -2).Range("A1").Select
    Application.CutCopyMode = False
    Selection.Copy
    ActiveCell.Offset(-2, 3).Range("A1").Select
    ActiveSheet.Paste
    ActiveCell.Offset(4, -3).Range("A1").Select
End Sub
```

NOTA Não estou sugerindo que o código anterior seja adequado para um aplicativo profissional. Mas, às vezes, só estamos escrevendo macros para automatizar uma única tarefa trivial.

Sem uma macro, eu teria executado manualmente várias vezes o comando copiar e colar. Agora, com a macro anterior gravada, eu posso posicionar o ponteiro de célula em um nome na Coluna A e pressionar Ctrl+Shift+A. Esse endereço será copiado para três colunas e o ponteiro de célula passará para o início do próximo endereço (veja Figura 5.13).

	A	B	C	D	E
1					
2	John Smith	John Smith	123 Main Street	Akron OH 44308	
3	123 Main Street				
4	Akron OH 44308				
5					
6	Jane Doe				
7	245 State Street				
8	Chicago IL 60011				
9					

Figura 5.13 Depois de executar a macro uma vez, um endereço é transformado no formato adequado e o ponteiro de célula é posicionado para executar a macro novamente.

Fiquei muito feliz com essa macro porque ela permitia processar um endereço por segundo utilizando a tecla de atalho. Percebi rapidamente que eu tinha 5 mil endereços e não queria executar repetidamente a mesma macro.

Utilizando um `Do...loop`, consegui configurar a macro para que ela executasse continuamente. Se incluísse o código gravado com `Do` na parte superior e o `Loop` no final, o VBA executaria contínua e repetidamente o código. Isso permitiria que eu relaxasse e observasse o código fazer o trabalho. Essa tarefa incrivelmente enfadonha, que levava horas, agora é feita em questão de minutos.

Observe que esse `Do...loop` em particular executará eternamente. Não há nenhum mecanismo para pará-lo. Isso funcionava para a tarefa à mão — eu poderia observar o progresso na tela e, depois que o programa ultrapassasse o final desse banco de dados, eu simplesmente pressionava Ctrl+Break para interromper a execução:

```
Sub Macro3()
'
' Macro Macro3
'macro gravada em 29/10/2003 por Bill Jelen
'move um endereço para o formato de banco de dados.
'move então o ponteiro de célula até o início do próximo endereço.
'
'atalhos pelo teclado: Ctrl+Shift+A
'
Do


    Selection.Copy
    ActiveCell.Offset(0, 1).Range("A1").Select
    ActiveSheet.Paste
    ActiveCell.Offset(1, -1).Range("A1").Select
    Application.CutCopyMode = False
    Selection.Copy
    ActiveCell.Offset(-1, 2).Range("A1").Select
    ActiveSheet.Paste
    ActiveCell.Offset(2, -2).Range("A1").Select
    Application.CutCopyMode = False
    Selection.Copy
```

```
        ActiveCell.Offset(-2, 3).Range("A1").Select
        ActiveSheet.Paste
        ActiveCell.Offset(4, -3).Range("A1").Select
    Loop
    End Sub
```

Todos esses exemplos são loops 'rápidos e sujos'. Eles são ótimos se você precisar realizar uma tarefa rapidamente. O Do...Loop fornece várias opções que permitem parar o programa automaticamente quando ele alcança o final da tarefa.

A primeira opção é fazer uma linha no `Do...loop` detectar o fim do conjunto de dados e sair do loop. No exemplo atual, isso poderia ser realizado utilizando o comando `Exit Do` em uma instrução `If`. Se a célula atual estiver em uma célula vazia, você pode supor que ele alcançou o fim dos dados e interrompeu o processamento do loop:

```
    Do
        If Not Selection.Value > "" Then Exit Do
        Selection.Copy
        ActiveCell.Offset(0, 1).Range("A1").Select
        ActiveSheet.Paste
        ActiveCell.Offset(1, -1).Range("A1").Select
        Application.CutCopyMode = False
        Selection.Copy
        ActiveCell.Offset(-1, 2).Range("A1").Select
        ActiveSheet.Paste
        ActiveCell.Offset(2, -2).Range("A1").Select
        Application.CutCopyMode = False
        Selection.Copy
        ActiveCell.Offset(-2, 3).Range("A1").Select
        ActiveSheet.Paste
        ActiveCell.Offset(4, -3).Range("A1").Select
    Loop
    End Sub
```

Utilizando a cláusula `While` ou `Until` no loop `Do`

Há quatro variações do uso de `While` ou `Until`. Essas cláusulas podem ser adicionadas à instrução `Do` ou `Loop`. Em cada caso, a cláusula `While` ou `Until` inclui algum teste que é avaliado como `True` ou `False`.

Com uma construção `Do While <expressão de teste>...loop`, o loop nunca é executado se `<expressão de teste>` for falso. Se estiver lendo registros a partir de um arquivo de texto, você não pode supor que o arquivo tenha um ou mais registros; portanto, é necessário testar para ver se você já está no fim do arquivo com a função `EOF` antes de entrar no loop:

```
    'lê um arquivo de texto, pulando as linhas Total
        Open "C:\fatura.txt" For Input As #1
        R = 1
        Do While Not EOF(1)
            Line Input #FileNumber, Data
            If Not Left (Data, 5) = "TOTAL" Then
                ' importa essa linha
                r = r + 1
                Cells(r, 1).Value = Data
            End If
        Loop
        Close #1
```

Nesse exemplo, usei a palavra-chave NOT. O EOF(1) é avaliado como True depois que não houver nenhum outro registro a ser lido em invoice.txt. Alguns programadores acreditam que é difícil ler um programa que contenha uma grande quantidade de NOTs. Para evitar o uso de NOT, utilize a construção `Do Until <test expression> ...Loop`:

```
    'lê um arquivo de texto, pulando as linhas Total
        Open "C:\fatura.txt" For Input As #1
        R = 1
        Do Until EOF(1)
            Line Input #1, Data
            If Not (Data, 5) = "TOTAL" Then
                ' importa essa linha
                r = r + 1
                Cells(r, 1).Value = Data
            End If
        Loop
        Close #1
```

Em outros exemplos, é recomendável executar o loop na primeira vez. Nesses casos, você move a instrução `While` ou `Until` para o fim do loop. Esse exemplo de código solicita que o usuário insira os valores das vendas feitas nesse dia. Ele socilita continuamente esses valores até o usuário inserir um zero:

```
TotalVendas = 0
Do
    x = InputBox(Prompt:="Insira a quantia da próxima fatura. Insira 0 ao terminar.")
    TotalVendas = TotalVendas + x
Loop Until x = 0
MsgBox "O total para hoje é $" & TotalVendas
```

No loop a seguir, um valor em cheque é inserido e, então, ele procura faturas em aberto (não pagas) às quais aplicar o cheque. É freqüente a situação em que um único cheque seja recebido para o pagamento de muitas faturas. Esse programa aplica seqüencialmente o cheque às faturas mais antigas até que 100 por cento do cheque tenha sido aplicado:

```
'solicita o valor do cheque recebido
AmtToApply = InputBox("Insira a Quantia do Cheque") + 0
'faz o loop pela lista de faturas abertas.
'aplica o cheque às faturas mais antigas e Decrementa AmtToApply
NextRow = 2
Do While AmtToApply > 0
    OpenAmt = Cells(NextRow, 3)
    If OpenAmt > AmtToApply Then
        ' aplica o cheque total a essa fatura
        Cells(NextRow, 4).Value = AmtToApply
        AmtToApply = 0
    Else
        Cells(NextRow, 4).Value = OpenAmt
        AmtToApply = AmtToApply - OpenAmt
    End If
    NextRow = NextRow + 1
Loop
```

Uma vez que você pode construir o Do...Loop com os qualificadores `While` ou `Until` no início ou no fim, você têm um controle bastante sutil sobre se o loop é sempre executado uma vez, mesmo se a condição for `True` no início.

Loops `While...Wend`

Os loops `While...Wend` foram incluídos no VBA para retrocompatibilidade. No arquivo de ajuda do VBA, a Microsoft sugere que os `Do...loops` são mais flexíveis. Mas como você poderia encontrar loops `While...Wend` no código escrito por outras pessoas, eis um exemplo rápido. Nesse loop, a primeira linha sempre é a condição `While`. A última linha do loop sempre é `Wend`. Observe que não há uma instrução `Exit While`. Em geral, esses loops são bons, mas a construção `Do...loop` é mais robusta e flexível. Como o loop `Do` oferece o qualificador `While` ou `Until`, esse qualificador pode estar no início ou no fim do loop e há a possibilidade de um loop `Do` fechar precocemente:

```
'lê um arquivo de texto, adicionando as quantidades
    Open "C:\fatura.txt" For Input As #1
    TotalVendas = 0
    While Not EOF(1)
        Line Input #1, Data
        TotalVendas = TotalVendas + Data
    Wend
    MsgBox "Total Vendas=" & TotalVendas
    Close #1
```

O loop do VBA: `For Each`

Esse é um excelente loop, e o programa de gravação de macros nunca grava loops desse tipo. O VBA é uma linguagem orientada a objetos. É comum existir uma coleção de objetos no Excel, como uma coleção de planilhas em uma *pasta de trabalho*, células em um intervalo, tabelas dinâmicas em uma planilha ou uma série de dados em um gráfico.

Esse tipo de loop especial é ótimo para fazer um loop por todos os itens na coleção. Antes de discutir esse loop em detalhes, você precisa entender um tipo especial de variável chamada *variáveis de objeto*.

Variáveis de objeto

Nesse ponto, você já viu uma variável que contém um único valor. Se tiver uma variável como `TotalVendas = 0`, `TotalVendas` é uma variável normal e geralmente contém apenas um valor único. Também é possível ter uma variável mais poderosa chamada *variável de objeto*. Uma variável de objeto armazena muitos valores; basicamente, todas as propriedades associadas ao objeto estão associadas à variável de objeto.

Em geral, não perco tempo para declarar minhas variáveis. Muitos livros imploram para que usemos a instrução `DIM` a fim de identificar todas as variáveis na parte superior do procedimento. Isso permite especificar que determinada variável é de certo tipo, como `Integer` ou `Double`. Embora isso economize um pouco de memória, também requer conhecer antecipadamente as variáveis que planejamos utilizar. Costumo inventar variáveis no decorrer do meu trabalho, criando uma nova variável instantaneamente conforme necessário. Mas há ótimos benefícios em declarar variáveis de objeto. O recurso AutoCompletar do VBA é ativado se você declarar uma variável de objeto na parte superior do seu procedimento. As linhas de código a seguir declaram três variáveis de objeto, uma como uma planilha, uma como um intervalo e uma como uma tabela dinâmica:

```
Sub Test()
    Dim WSD as Worksheet
    Dim MyCell as Range
    Dim PT as PivotTable
    Set WSD = ThisWorkbook.Worksheets("Dados")
    Set MyCell = WSD.Cells(Rows.Count, 1).End(xlUp).Offset(1, 0)
    Set PT = WSD.PivotTables(1)
    ...
```

No exemplo anterior, você pode ver que apenas uma instrução equals não é utilizada para atribuir variáveis de objeto. Você precisa utilizar a instrução `Set` para atribuir um objeto específico à variável de objeto.

Há muitas boas razões de se utilizar variáveis de objeto, uma delas, não menos importante, é o fato de que isso pode ser uma ótima notação de abreviação. É muito mais fácil fazer uma grande quantidade de linhas de código referirem-se a `WSD` em vez de `ThisWorkbook.Worksheets("Dados")`.

Além disso, como já mencionado, a variável de objeto herda todas as propriedades do objeto a que se refere.

O `For Each...Loop` emprega uma variável de objeto em vez de uma variável `Counter`. O próximo código faz um loop por todas as células na Coluna A. O código utiliza a propriedade `.CurrentRegion` para definir a região atual e, então, utiliza a propriedade `.Resize` para limitar o intervalo selecionado a uma única coluna. A variável de objeto é chamada `Cell`. Eu poderia ter usado qualquer nome para a variável de objeto, mas `Cell` parece mais apropriado do que algo arbitrário como Fred:

```
For Each cell in Range("A1").CurrentRegion.Resize(, 1)
    If cell.Value = "Total" Then
        cell.resize(1,8).Font.Bold = True
    End If
Next cell
```

Esse exemplo de código pesquisa todas as pastas de trabalho abertas, procurando um nome particular de planilha:

```
For Each wb in Workbooks
    If wb.Worksheets(1).Name = "Menu" Then
    WBFound = True
    WBName = wb.Name
    Exit For
End If
Next wb
```

Nesse exemplo de código, todas as formas na planilha atual são excluídas:

```
For Each Sh in ActiveSheet.Shapes
    Sh.Delete
Next Sh
```

Esse exemplo de código exclui todas as tabelas dinâmicas na planilha atual:

```
For Each pt in ActiveSheet.PivotTables
    pt.TableRange2.Clear
Next pt
```

Estudo de caso

Fazendo um loop por todos os arquivos em um diretório

Eis alguns procedimentos úteis que utilizam extensamente loops.

> NOTA
>
> Criar uma lista de todos os arquivos em um diretório costumava ser relativamente simples usando o objeto `FileSearch`. Por razões inexplicáveis, a Microsoft parou de suportar `FileSearch` no Excel 2007.

O primeiro procedimento utiliza o objeto `FileSearch` do VBA para localizar todos os arquivos de imagens JPG em um certo diretório. No Excel, todos os arquivos são listados em uma coluna.

O loop `For i` externo itera por cada arquivo de imagem encontrado em uma determinada pasta e em todas as suas subpastas. A cada passagem, o loop externo retornará um novo caminho e nome de arquivo na variável `ThisEntry`.

Para separar o caminho do nome de arquivo, o loop `For j` interno pesquisará o caminho e o nome de arquivo do fim ao início, procurando pelo separador de caminho final:

```
Sub FindJPGFilesInAFolder()
    Dim fso As Object
    Dim strName As String
    Dim strArr(1 To 1048576, 1 To 1) As String, i As Long

    ' insira o nome da pasta aqui
    Const strDir As String = "C:\Artwork\"

    Let strName = Dir$(strDir & "*.jpg")
    Do While strName <> vbNullString
        Let i = i + 1
        Let strArr(i, 1) = strDir & strName
        Let strName = Dir$()
    Loop
    Set fso = CreateObject("Scripting.FileSystemObject")
    Call recurseSubFolders(fso.GetFolder(strDir), strArr(), i)
    Set fso = Nothing
    If i > 0 Then
        Range("A1").Resize(i).Value = strArr
    End If

    ' em seguida, faz um loop por todos os arquivos localizados e divide o caminho e nome de arquivo
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    For i = 1 To FinalRow
        ThisEntry = Cells(i, 1)
        For j = Len(ThisEntry) To 1 Step -1
            If Mid(ThisEntry, j, 1) = Application.PathSeparator Then
                Cells(i, 2) = Left(ThisEntry, j)
                Cells(i, 3) = Mid(ThisEntry, j + 1)
                Exit For
            End If
        Next j
    Next i

End Sub

Private Sub recurseSubFolders(ByRef Folder As Object, _
    ByRef strArr() As String, _
    ByRef i As Long)
Dim SubFolder As Object
Dim strName As String
For Each SubFolder In Folder.SubFolders
    Let strName = Dir$(SubFolder.Path & "*.jpg")
    Do While strName <> vbNullString
        Let i = i + 1
        Let strArr(i, 1) = SubFolder.Path & strName
        Let strName = Dir$()
    Loop
    Call recurseSubFolders(SubFolder, strArr(), i)
Next
End Sub
```

Minha idéia, nesse caso, é que quero organizar as fotos em pastas novas. Na Coluna D, se eu quiser mover uma imagem para uma nova pasta, digito o caminho dessa pasta. O próximo loop `For Each` trata de copiar as imagens. Toda vez que passar pelo loop, a variável de objeto chamada `Cell` conterá uma referência a uma célula na Coluna A. Você pode utilizar `Cell.Offset(0, 3)` para retornar o valor da célula três colunas à direita do intervalo representado pela variável `Cell`:

```
Sub CopyToNewFolder()
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    For Each Cell In Range("A2:A" & FinalRow)
        OrigFile = Cell.Value
        NewFile = Cell.Offset(0, 3) & Application.PathSeparator & _
            Cell.Offset(0, 2)
        FileCopy OrigFile, NewFile
    Next Cell
End Sub
```

Controle de fluxo: Usando `If...Then...Else` e `Select Case`

Outro aspecto da programação que nunca será gravado pelo programa de gravação de macros é o conceito de controle de fluxo. Às vezes, você não quer que cada linha de seu programa seja executada toda vez que usa a macro. O VBA oferece duas escolhas excelentes para o controle de fluxo: a construção `If...Then...Else`, e a construção `Select Case`.

Controle de fluxo básico: `If...Then...Else`

O dispositivo mais comum para controle de fluxo de programa é a instrução `If`. Suponha, por exemplo, que você tenha uma lista de produtos como mostrado na Figura 5.14. Você quer fazer um loop por cada produto na lista e copiá-lo para uma lista de Frutas ou Verduras. Como programador iniciante, eu era tentado a fazer um loop pelas linhas duas vezes: eu queria fazer um loop uma vez procurando frutas e uma segunda vez procurando verduras. Mas não há necessidade disso. Em um único loop, você pode utilizar uma construção `If...Then...Else` para copiar cada linha para o lugar correto.

Figura 5.14
Um único loop pode procurar por frutas ou legumes.

	A	B	C	D
1	Classe	Produto	Quantidade	
2	Frutas	Maçãs	1	
3	Frutas	Damasco	3	
4	Legumes	Aspargos	62	
5	Frutas	Bananas	55	
6	Frutas	Arando	17	
7	Legumes	Brócolis	56	
8	Legumes	Repolho	35	
9	Frutas	Cerejas	59	
10	Temperos	Condimento	91	
11	Legumes	Berinjela	94	
12	Frutas	Kiwi	86	

Condições

Qualquer instrução `If` precisa de uma condição que esteja sendo testada. A condição sempre deve ser avaliada como `TRUE` ou `FALSE`. Eis alguns exemplos de condições simples e complexas:

- `If Range("A1").Value = "Title" Then`
- `If Not Range("A1").Value = "Title" Then`
- `If Range("A1").Value = "Title" And Range("B1").Value = "Frutas" Then`
- `If Range("A1").Value = "Title" Or Range("B1").Value = "Frutas" Then`

`If...Then...End If`

Depois da instrução `If`, você pode incluir uma ou mais linhas de programa que serão executadas somente se a condição for atendida. Você então deve fechar o bloco `If` com uma linha `End If`. Eis um exemplo simples de uma instrução `If`:

```
Sub ColorFruitRedBold()
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row

    For i = 2 To FinalRow
        If Cells(i, 1).Value = "Frutas" Then
            Cells(i, 1).Resize(1, 3).Font.Bold = True
            Cells(i, 1).Resize(1, 3).Font.ColorIndex = 3
        End If
    Next i

    MsgBox "Frutas agora estão em negrito e vermelho"
End Sub
```

Decisões Either/Or: If...Then...Else...End If

Às vezes, você irá querer criar um conjunto de instruções se a condição for verdadeira e outro conjunto se a condição não for verdadeira. Para fazer isso com o VBA, o segundo conjunto de condições seria codificado depois da instrução `Else`. Ainda há uma única instrução `End If` associada a essa construção. Suponha, por exemplo, que você queira aplicar vermelho a frutas e verde a legumes:

```
Sub FruitRedVegGreen()
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row

    For i = 2 To FinalRow
        If Cells(i, 1).Value = "Frutas" Then
            Cells(i, 1).Resize(1, 3).Font.ColorIndex = 3
        Else
            Cells(i, 1).Resize(1, 3).Font.ColorIndex = 50
        End If
    Next i

    MsgBox "Frutas estão em vermelho / Legumes estão em verde"
End Sub
```

Utilizando If...Else If...End If para múltiplas condições

Note que nossa lista de produtos inclui um item que é classificado como tempero. Na verdade, temos três condições em que fazer o teste. É possível construir uma estrutura `If...End If` com múltiplas condições. Primeiro, teste para ver se o registro é uma fruta. Em seguida, utilize uma `Else If` para testar se o registro é um legume. Então, teste para ver se o registro é uma erva. Por fim, se o registro não for nenhum desses, destaque-o como erro:

```
Sub MultipleIf()
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row

    For i = 2 To FinalRow
        If Cells(i, 1).Value = "Frutas" Then
            Cells(i, 1).Resize(1, 3).Font.ColorIndex = 3
        ElseIf Cells(i, 1).Value = "Legumes" Then
            Cells(i, 1).Resize(1, 3).Font.ColorIndex = 50
        ElseIf Cells(i, 1).Value = "Temperos" Then
            Cells(i, 1).Resize(1, 3).Font.ColorIndex = 5
        Else
            ' isto deve ser um registro no erro
            Cells(i, 1).Resize(1, 3).Interior.ColorIndex = 6
        End If
    Next i

    MsgBox "Frutas estão em vermelho / Legumes estão em verde / Temperos estão em azul"
End Sub
```

Utilizando Select Case...End Select para múltiplas condições

Quando você tem muitas condições diferentes, torna-se difícil controlar o uso de várias instruções `Else If`. O VBA oferece outra construção conhecida como a construção `Select Case`. Na nossa execução de exemplo, sempre queremos verificar o valor da Classe na coluna A. Esse valor é chamado *expressão de teste*. A sintaxe básica dessa construção inicia com as palavras `Select Case` seguidas pela expressão de teste:

```
Select Case Cells(i, 1).Value
```

Pensando no nosso problema em inglês, você poderia dizer: "Nos casos em que o registro é fruta, aplique a cor vermelha ao registro". O VBA utiliza uma versão abreviada disso. Você escreve a palavra `Case` seguida do literal "`Frutas`". Qualquer instrução depois de `Case "Frutas"` será executada sempre que a expressão de teste for uma fruta. Depois dessas instruções, você teria a próxima instrução `Case`: `Case "Legumes"`. Você continuaria dessa maneira, escrevendo uma instrução `Case` seguida pelas linhas de programa que serão executadas se esse caso for verdadeiro.

Depois de ter listado todas as possíveis condições em que poderia pensar, inclua, opcionalmente, uma seção `Case Else` no final. Essa seção inclui o que o programa deve fazer se a expressão de teste não corresponder a nenhum dos casos. Por fim, feche a construção inteira com a instrução `End Select`.

O código a seguir faz a mesma operação que a macro anterior, mas utiliza uma instrução `Select Case`:

```
Sub SelectCase()
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row

    For i = 2 To FinalRow
        Select Case Cells(i, 1).Value
            Case "Frutas"
                Cells(i, 1).Resize(1, 3).Font.ColorIndex = 3
            Case "Legumes"
                Cells(i, 1).Resize(1, 3).Font.ColorIndex = 50
            Case "Temperos"
                Cells(i, 1).Resize(1, 3).Font.ColorIndex = 5
            Case Else
        End Select
    Next i

    MsgBox "Frutas estão em vermelho / Legumes estão em verde / Temperos estão em azul"
End Sub
```

Expressões complexas em instruções Case

É possível ter expressões relativamente complexas em instruções `Case`. Talvez você queira realizar as mesmas ações para todos os registros de frutas:

```
Case "Morango", "Arando", "Framboesa"
     AdCode = 1
```

Se isso fizer sentido, você poderá codificar um intervalo de valores na instrução `Case`:

```
Case 1 to 20
    Discount = 0.05
Case 21 to 100
    Discount = 0.1
```

Você pode incluir a palavra-chave `Is` e um operador de comparação, como > ou <:

```
Case Is < 10
    Discount = 0
Case Is > 100
    Discount = 0.2
Case Else
    Discount = 0.10
```

Aninhando instruções `If`

É possível e comum aninhar uma instrução `If` dentro de outra instrução `If`. Nessa situação, é muito importante utilizar o recuo adequado. Você verá que muitas vezes há várias linhas `End If` no final da construção. Utilizando um recuo adequado, é mais fácil dizer que `End If` está associada a uma `If` determinada.

A macro final tem muita lógica. Nossas regras de desconto são as seguintes:

- Para frutas, quantidades abaixo de 5 caixas não têm desconto.
- Quantidades entre 5 e 20 caixas têm um desconto de 10%.
- Quantidades acima de 20 caixas têm um desconto de 15%.
- Para temperos, quantidades abaixo de 10 caixas não têm nenhum desconto.
- Quantidades entre 10 e 15 caixas têm um desconto de 3%.
- Quantidades acima de 15 caixas têm um desconto de 6%.
- Para legumes, exceto aspargos, 5 ou mais caixas têm um desconto de 12%.
- O aspargo requer 20 caixas para que tenha um desconto de 12%.
- Nenhum dos descontos é aplicado se o produto estiver em promoção nessa semana. O preço de venda tem um desconto de 25% em relação ao preço normal. Os itens à venda nessa semana são morango, alface e tomate.

O código para executar essa lógica é mostrado a seguir:

```
Sub ComplexIf()
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row

    For i = 2 To FinalRow
        ThisClass = Cells(i, 1).Value
        ThisProduct = Cells(i, 2).Value
        ThisQty = Cells(i, 3).Value
        ' primeiro, descobre se o item está em promoção
        Select Case ThisProduct
            Case "Morango", "Legumes", "Tomates"
                Sale = True
            Case Else
                Sale = False
        End Select

        ' descobre o desconto
        If Sale Then
            Discount = 0.25
        Else
            If ThisClass = "Frutas" Then
                Select Case ThisQty
                    Case Is < 5
                        Discount = 0
                    Case 5 To 20
                        Discount = 0.1
                    Case Is > 20
                        Discount = 0.15
                End Select
            ElseIf ThisClass = "Temperos" Then
                Select Case ThisQty
                    Case Is < 10
                        Discount = 0
                    Case 10 To 15
                        Discount = 0.03
                    Case Is > 15
                        Discount = 0.05
                End Select
            ElseIf ThisClass = "Legumes" Then
                ' há uma condição especial para aspargo
                If ThisProduct = "Aspargos" Then
                    If ThisQty < 20 Then
                        Discount = 0
                    Else
                        Discount = 0.12
                    End If
                Else
                    If ThisQty < 5 Then
                        Discount = 0
                    Else
                        Discount = 0.12
                    End If
                End If ' o produto é ou não aspargos?
            End If ' o produto é um legume?
        End If ' o produto está em promoção?

        Cells(i, 4).Value = Discount

        If Sale Then
            Cells(i, 4).Font.Bold = True
        End If

    Next i

    Range("D1").Value = "Discount"

    MsgBox "Os descontos foram aplicados"

End Sub
```

Próximos passos

Loops dão um poder enorme às macros gravadas. Sempre que você precisar repetir um processo em todas as planilhas ou em todas as linhas de uma planilha, um loop será a opção a usar. O Excel VBA suporta os tradicionais loops de programação `For...Next` e `Do...loop`, bem como o loop orientado a objetos `For Each...Next`. O Capítulo 6, "Fórmulas no estilo L1C1", discute o estilo L1C1 aparentemente misterioso de fórmulas e mostra por que ele é importante no Excel VBA.

Fórmulas no estilo L1C1

6

Referenciando células: referências A1 *versus* L1C1

Podemos identificar retroativamente o estilo A1 de referenciar no VisiCalc. Dan Bricklin e Bob Frankston utilizaram A1 para se referirem à célula no canto superior esquerdo da planilha. Mitch Kapor utilizou esse mesmo esquema de endereçamento no Lotus 1-2-3. Upstart Multiplan, da Microsoft, tentou resistir e usou algo chamado 'endereçamento no estilo L1C1'. No endereçamento L1C1, a célula conhecida como A1 é referida como L1C1 porque está na Linha 1 e na Coluna 1.

Com o domínio do Lotus 1-2-3 na década de 1980 e começo dos anos 1990, o estilo A1 tornou-se o padrão. A Microsoft percebeu que lutava uma batalha perdida e, por fim, ofereceu o endereçamento no estilo L1C1 ou endereçamento no estilo A1 no Excel. Quando você abre o Excel hoje, o estilo A1 é utilizado por padrão. Oficialmente, porém, a Microsoft suporta os dois estilos de endereçamento.

Você poderia achar que este capítulo traria uma questão sem importância. Qualquer pessoa que utiliza a interface Excel concordaria que o estilo L1C1 está morto. Entretanto, temos o que aparentemente seria um problema irritante: o programa de gravação de macros grava fórmulas no estilo L1C1. Portanto, talvez você esteja pensando que só terá de aprender o endereçamento L1C1 para ler o código gravado e voltar ao estilo A1 familiar.

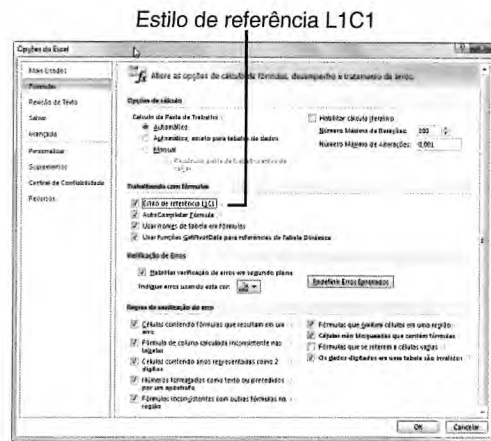
Preciso dar créditos à Microsoft. Depois de entender fórmulas no estilo L1C1, você, na verdade, descobrirá que elas são mais eficientes, especialmente para escrever fórmulas no VBA. Utilizar o endereçamento no estilo L1C1 permite escrever código mais eficientemente. Além disso, há alguns recursos, como configurar fórmulas de array ou formatação condicional com a opção `Formula Is`, em que você é solicitado a inserir a fórmula no estilo L1C1.

Posso ouvir a lamuria coletiva dos usuários do Excel em todos os lugares. Eu faria tudo para pular as dez páginas desse estilo de endereçamento antiquado se ele fosse apenas um aborrecimento ou uma questão de eficiência. Entretanto, como é necessário entender o endereçamento L1C1 para usar efetivamente recursos importantes como fórmulas de array ou formatação condicional, temos de nos aprofundar e aprender esse estilo.

Mudando o Excel para exibir referências no estilo L1C1

Para mudar para o endereçamento no estilo L1C1, selecione Opções do Excel no ícone Office. Na categoria Fórmulas, marque a caixa de estilo de referência L1C1 (veja Figura 6.1).

NESTE CAPÍTULO

Figura 6.1
Marcar o estilo de referência L1C1 na guia Geral da caixa Opções faz com que o Excel reverta para o estilo L1C1 na interface com o usuário.

Depois de mudar para o estilo L1C1, as letras da coluna A, B, C na parte superior da planilha são substituídas pelos números 1, 2, 3 (veja Figura 6.2).

L5C2 | =ARRED(L1C2*LC[-1];2)

	1	2	3	4	5	6
1	Taxa de impostos	6,25%				
2						
3						
4	Venda	Impostos	Total			
5	125	7,81	132,8			
6						
7						
8						
9						

Figura 6.2
No estilo L1C1, as letras da coluna são substituídas por números.

Nesse formato, a célula que você conhece como B5 é chamada L5C2, porque está na Linha 5, Coluna 2.

Mais ou menos a cada duas semanas, alguém ativa acidentalmente essa opção e recebemos uma solicitação urgente de suporte no MrExcel; o estilo é muito estranho para 99% dos usuários de planilha.

O milagre das fórmulas do Excel

Recalcular automaticamente milhares de células é o principal benefício das planilhas eletrônicas em relação ao livro-razão utilizado até 1979. Mas eu diria que uma segunda vantagem, também importante, seria o fato de que é possível inserir uma fórmula e copiá-la para milhares de células.

Insira uma fórmula uma vez e a copie 1.000 vezes

Considere essa planilha simples na Figura 6.3. Insira uma fórmula simples como `=C4*B4` na Célula D4, dê um clique duplo na alça Preenchimento e a fórmula mudará inteligentemente à medida que é copiada para o intervalo.

Botão 1

	A	B	C	D	E	F	G
1	Taxa de impostos	6,25%					
2							
3	Código	Quantidade	Preço Unitário	Preço Total	Taxável?	Taxa	Total
4	217	12	12,45	149,4	VERDADEIRO	9,34	
5	123	144	1,87		VERDADEIRO		
6	329	18	19,95		VERDADEIRO		
7	616	1	642		FALSO		
8	909	64	17,5		VERDADEIRO		
9	527	822	0,12		VERDADEIRO		
10	Total						
11							
12							

Figura 6.3
Dê um clique duplo na alça Preenchimento e o Excel copia de modo inteligente essa fórmula de referência relativa para a coluna.

A fórmula em F4 inclui tanto fórmulas relativas como absolutas: `=SE(E4,ARRED(D4*$B$1,2),0)`. Graças aos sinais de cifrão inseridos em B1, você pode copiar essa fórmula e ela sempre multiplicará o Preço Total nessa linha pela alíquota de imposto na Célula B1.

Os resultados numéricos na Figura 6.4 são alcançados pelas fórmulas mostradas na Figura 6.5. (Ctrl+~ no Excel alterna entre a visualização Normal e a visualização Fórmula.) Considerando que eu tinha de inserir fórmulas apenas nas Linhas 4 e 10, achei surpreendente o fato de o Excel ser capaz de copiar inteligentemente as fórmulas para a coluna.

Como usuários do Excel, tomamos esse comportamento como certo, mas as pessoas nas aulas de Excel para iniciantes se surpreendem com o fato de que a fórmula `=F4+D4` na célula G4 muda automaticamente para `=F5+D5` quando é copiada para a célula G5.

Figura 6.4
Esses resultados nas colunas D, F e G são alcançados pelas fórmulas mostradas na Figura 6.5.

	A	B	C	D	E	F	G	H
1	Taxa de impostos	6,25%						
2								
3	Código	Quantidade	Preço Unitário	Preço Total	Taxável?	Taxa	Total	
4	217	12	12,45	149,4	VERDADEIRO	9,34	158,74	
5	123	144	1,87	269,28	VERDADEIRO	16,83	286,11	
6	329	18	19,95	359,1	VERDADEIRO	22,44	381,54	
7	616	1	642	642	FALSO	0	642,00	
8	909	64	17,5	1120	VERDADEIRO	70	1190,00	
9	527	822	0,12	98,64	VERDADEIRO	6,17	104,81	
10	Total						2763,20	
11								

Figura 6.5
Pressione Ctrl+~ para mostrar fórmulas em vez dos resultados. É surpreendente o fato de o Excel ajustar as referências de célula em cada fórmula enquanto você copia para a coluna.

	A	B	C	D	E	F	G
1	Taxa de impostos	0,0625					
2							
3	Código	Quantidade	Preço Unitário	Preço Total	Taxável?	Taxa	Total
4	217	12	12,45	=B4*C4	VERDADEIRO	=SE(E4;ARRED(D4*$B$1;2);0)	=F4+D4
5	123	144	1,87	=B5*C5	VERDADEIRO	=SE(E5;ARRED(D5*$B$1;2);0)	=F5+D5
6	329	18	19,95	=B6*C6	VERDADEIRO	=SE(E6;ARRED(D6*$B$1;2);0)	=F6+D6
7	616	1	642	=B7*C7	FALSO	=SE(E7;ARRED(D7*$B$1;2);0)	=F7+D7
8	909	64	17,5	=B8*C8	VERDADEIRO	=SE(E8;ARRED(D8*$B$1;2);0)	=F8+D8
9	527	822	0,12	=B9*C9	VERDADEIRO	=SE(E9;ARRED(D9*$B$1;2);0)	=F9+D9
10	Total						=SOMA(G4:G9)
11							

O segredo — não é tão surpreendente

Lembre-se de que o Excel faz tudo nas fórmulas no estilo L1C1. O Excel mostra endereços e fórmulas no estilo A1 apenas porque precisa estar em conformidade com o padrão popularizado pelo VisiCalc e Lotus.

Se mudar a planilha na Figura 6.5 para que use a notação L1C1, você notará que as "diferentes" fórmulas em D4:D9 são, na verdade, fórmulas idênticas na notação L1C1. O mesmo é verdade para F4:F9 e G4:G9. Utilize a caixa de diálogo Opções para mudar a planilha de exemplo para endereços no estilo L1C1. Se examinar as fórmulas na Figura 6.6, verá que na linguagem L1C1 todas as fórmulas na Coluna D são idênticas. Dado que o Excel armazena as fórmulas no estilo L1C1, as copia e então simplesmente as converte no estilo A1 para que possamos entender, deixa de ser tão surpreendente que o Excel possa manipular facilmente as fórmulas no estilo A1 como faz.

Figura 6.6
As mesmas fórmulas no estilo L1C1. Observe que todas as fórmulas na Coluna 4 ou 6 são idênticas a todas as outras fórmulas nessa coluna.

	1	2	3	4	5	6	7
1	Taxa de impostos	0,0625					
2							
3	Código	Quantidade	Preço Unitário	Preço Total	Taxável?	Taxa	Total
4	217	12	12,45	=LC[-2]*LC[-1]	VERDADEIRO	=SE(LC[-1];ARRED(LC[-2]*L1C2;2);0)	=LC[-1]+LC[-3]
5	123	144	1,87	=LC[-2]*LC[-1]	VERDADEIRO	=SE(LC[-1];ARRED(LC[-2]*L1C2;2);0)	=LC[-1]+LC[-3]
6	329	18	19,95	=LC[-2]*LC[-1]	VERDADEIRO	=SE(LC[-1];ARRED(LC[-2]*L1C2;2);0)	=LC[-1]+LC[-3]
7	616	1	642	=LC[-2]*LC[-1]	FALSO	=SE(LC[-1];ARRED(LC[-2]*L1C2;2);0)	=LC[-1]+LC[-3]
8	909	64	17,5	=LC[-2]*LC[-1]	VERDADEIRO	=SE(LC[-1];ARRED(LC[-2]*L1C2;2);0)	=LC[-1]+LC[-3]
9	527	822	0,12	=LC[-2]*LC[-1]	VERDADEIRO	=SE(LC[-1];ARRED(LC[-2]*L1C2;2);0)	=LC[-1]+LC[-3]
10	Total						=SOMA(L[-6]C:L[-1]C)
11							

Essa é uma das razões de as fórmulas no estilo L1C1 serem mais eficientes no VBA. Você pode inserir a mesma fórmula em um intervalo inteiro de dados em uma única instrução.

Estudo de caso

Inserindo A1 versus L1C1 no VBA

Pense como você definiria essa planilha na interface do Excel. Primeiro, você insere uma fórmula nas células D4, F4, G4. Em seguida, copia essas células e, depois, as cola na coluna. O código poderia ser parecido com isto:

```
Sub A1Style()
    ' Localiza o FinalRow
    FinalRow = Cells(Rows.Count, 2).End(xlUp).Row
    ' Insere a primeira fórmula
    Range("D4").Formula = "=B4*C4"
    Range("F4").Formula = "=SE(E4,ARRED(D4*$B$1,2),0)"
    Range("G4").Formula = "=F4+D4"
    ' Copia as fórmulas da Linha 4 para as outras células
    Range("D4").Copy Destination:=Range("D5:D" & FinalRow)
    Range("F4:G4").Copy Destination:=Range("F5:G" & FinalRow)
    ' Insere a linha Total
    Cells(FinalRow + 1, 1).Value = "Total"
    Cells(FinalRow + 1, 6).Formula = "=SOMA(G4:G" & FinalRow & ")"
End Sub
```

Nesse código, são necessárias três linhas para inserir as fórmulas no começo da linha e outras duas linhas para copiar as fórmulas para a coluna.

O código equivalente no estilo L1C1 permite que as fórmulas sejam inseridas para a coluna inteira por meio de uma única instrução. Lembre-se de que a vantagem das fórmulas no estilo L1C1 é que todas as fórmulas nas colunas D, F e na maior parte de G são idênticas:

```
Sub L1C1Style()
    ' Localiza o FinalRow
    FinalRow = Cells(Rows.Count, 2).End(xlUp).Row
    ' Insere a primeira fórmula
    Range("D4:D" & FinalRow).FormulaL1C1 = "=LC[-1]*LC[-2]"
    Range("F4:F" & FinalRow).FormulaL1C1 = "=SE(LC[-1],ARRED(LC[-2]*R1C2,2),0)"
    Range("G4:G" & FinalRow).FormulaL1C1 = "=+LC[-1]+LC[-3]"
    ' Insere a linha Total
    Cells(FinalRow + 1, 1).Value = "Total"
    Cells(FinalRow + 1, 6).Formula = "=SOMA(G4:G" & FinalRow & ")"
End Sub
```

Explicação do estilo de referência L1C1

Uma referência no estilo L1C1 inclui a letra *L* para referenciar a linha e a letra *C* para referenciar a coluna. Como a referência mais comum em uma fórmula é uma referência relativa, vejamos primeiro estas no estilo L1C1.

Utilizando L1C1 com referências relativas

Imagine que você esteja inserindo uma fórmula em uma célula. Para apontar para uma célula em uma fórmula, utilize as letras `L` e `C`. Depois de cada letra, insira o número de linhas ou colunas entre colchetes.

A próxima lista explica as 'regras' para utilizar referências relativas em L1C1:

- Para as colunas, um número positivo significa mover-se para a direita um determinado número de colunas e um número negativo significa mover-se para esquerda um determinado número de colunas. A partir da célula E5, utilize `LC[1]` para referenciar F5 e `LC[-1]` para referenciar D5.
- Para as linhas, um número positivo significa mover a planilha para baixo na um determinado número de linhas. Um número negativo significa mover-se para o começo da planilha um determinado número de linhas. A partir da célula E5, utilize `L[1]C` para referenciar E6 e use a célula `L[-1]C` para referenciar E4.
- Se omitir os colchetes para `L` ou para `C`, significará que está apontando para uma célula na mesma linha ou coluna que a célula com a fórmula.
- Se inserir `=L[-1]C[-1]` na célula E5, estará se referindo a uma célula uma linha acima e uma coluna à esquerda. Essa seria a célula D4.
- Se inserir `=LC[-1]` na célula E5, estará referenciando uma célula na mesma linha, mas uma coluna à esquerda. Essa seria a célula D5.
- Se inserir `=LC[1]` na célula E5, estará referenciando uma célula na mesma linha, mas uma coluna à direita. Essa seria a célula F5.
- Se inserir `=LC` na célula E5, estará referenciando uma célula na mesma linha e coluna, que é a própria célula E5. Em geral, nunca faça isso porque você criaria uma referência circular.

A Figura 6.7 mostra como inserir uma referência na célula E5 para apontar para várias células em torno de E5.

Figura 6.7
Eis várias referências relativas. Elas seriam inseridas na célula E5 para descrever cada célula em torno de E5.

Você pode utilizar o estilo L1C1 para referenciar um intervalo de células. Se quiser somar as 12 *células* à esquerda da célula atual, a fórmula é:

```
=SOMA(LC[-12]:LC[-1])
```

Utilizando L1C1 com referências absolutas

Uma referência absoluta é aquela em que a linha e a coluna permanecem fixas quando a fórmula é copiada para um novo local. Na notação no estilo A1, o Excel utiliza um `$` antes do número da linha ou da letra da coluna para manter tanto essa linha como a coluna absolutas enquanto a fórmula é copiada.

Para referenciar sempre um número absoluto de linha ou coluna, apenas omita os colchetes. Essa referência está relacionada à célula $B$2, independentemente de onde ela é inserida:

```
=L2C2
```

Utilizando L1C1 com referências mistas

Uma referência mista é aquela em que a linha é fixa e a coluna pode ser relativa ou em que a coluna é fixa e a linha pode ser relativa. Em muitas situações, isso será útil.

Imagine que você escreveu uma macro para importar Invoice.txt para o Excel. Utilizando `.End(xlUp)`, você localiza onde posicionar a linha total. Como está inserindo totais, sabe que quer somar a partir da linha acima da fórmula até a linha 2. O próximo código trataria isso:

```
Sub MixedReference()
    TotalRow = Cells(Rows.Count, 1).End(xlUp).Row + 1
    Cells(TotalRow, 1).Value = "Total"
    Cells(TotalRow, 5).Resize(1, 3).FormulaL1C1 = "=SOMA(L2C:L[-1]C)"
End Sub
```

Nesse código, a referência `L2C:L[1]C` indica que a fórmula deve ser adicionada da Linha 2 na mesma coluna à linha logo acima da fórmula na coluna atual. Você vê a vantagem das fórmulas L1C1 nesse caso? Uma única fórmula L1C1 com uma referência mista pode ser facilmente utilizada para inserir uma fórmula a fim de tratar um número indeterminado de linhas de dados (veja Figura 6.8).

Figura 6.8
Depois de executar a macro, as fórmulas na coluna E:G da linha Total terão referência a um intervalo que é bloqueado na linha 2, mas todos os outros aspectos são relativos.

PGTO =SOMA(G$2:G13)

	A	B	C	D	E	F	G	H
1	DataFatura	NumeroFat	NumeroVend	NumeroClient	ReceitaProd	ReceitaServi	CustoProduto	
2	05/06/2004	123801	S82	C8754	639600	12000	325438	
3	05/06/2004	123802	S93	C7874	964600	0	435587	
4	05/06/2004	123803	S43	C4844	988900	0	587630	
5	05/06/2004	123804	S54	C4940	673800	15000	346164	
6	05/06/2004	123805	S43	C7969	513500	0	233842	
7	05/06/2004	123806	S93	C8468	760600	0	355305	
8	05/06/2004	123807	S82	C1620	894100	0	457577	
9	05/06/2004	123808	S17	C3238	316200	45000	161877	
10	05/06/2004	123809	S32	C5214	111500	0	62956	
11	05/06/2004	123810	S45	C3717	747600	0	444162	
12	05/06/2004	123811	S87	C7492	857400	0	410493	
13	05/06/2004	123812	S43	C7780	200700	0	97937	
14	Total				7668500	72000	=SOMA(G$2:G13)	
15								

Referindo-se a colunas ou linhas inteiras com o estilo L1C1

De vez em quando, você escreverá uma fórmula que referencia uma coluna inteira. Por exemplo, talvez você queira conhecer o valor máximo na Coluna G. Se não souber quantas linhas G terá, você poderá escrever `=MAX($G:$G)` no estilo A1 ou `=MAX(C7)` no L1C1. Para localizar o valor mínimo na Linha 1, utilize `=MIN($1:$1)` no estilo A1 ou `=MIN(L1)` no L1C1. Você pode utilizar uma referência relativa para linhas ou colunas. Para localizar a média da célula acima da linha atual, utilize `=AVERAGE(L[-1])`.

Substituindo muitas fórmulas A1 por uma única fórmula L1C1

Depois de se acostumar com as fórmulas no estilo L1C1, realmente parece muito mais intuitivo construí-las. Um exemplo clássico para ilustrar fórmulas no estilo L1C1 é construir uma tabela de multiplicação. É fácil criar uma tabela de multiplicação no Excel com uma única fórmula de referência mista.

Construindo a tabela

Insira os números `1` a `12` abrangendo B1:M1. Copie e transponha esses números para que eles abranjam A2:A13. Agora o desafio é construir uma única fórmula que funcione em todas as células de B2:M13 e que mostre a multiplicação do número na Linha 1 pelo número na Coluna 1. Utilizando fórmulas no estilo A1, você deve pressionar cinco vezes a tecla F4 para obter sinais de cifrão nos locais adequados. A seguir, é mostrada uma fórmula muito mais simples no estilo L1C1:

```
Sub MultiplicationTable()
    ' Construir uma tabela de multiplicação utilizando uma única fórmula
    Range("B1:M1").Value = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
    Range("B1:M1").Font.Bold = True
    Range("B1:M1").Copy
```

```
    Range("A2:A13").PasteSpecial Transpose:=True
    Range("B2:M13").FormulaL1C1 = "=LC1*L1C"
    Cells.EntireColumn.AutoFit
End Sub
```

A referência no estilo L1C1 `=LC1*L1C` não poderia ser mais simples. No idioma inglês, isso significa, 'pegue a Coluna 1 dessa linha e multiplique-a pela Linha 1 dessa coluna'. Funciona perfeitamente para construir a tabela de multiplicação mostrada na Figura 6.9.

Figura 6.9
A macro cria uma tabela de multiplicação. A fórmula em B2 utiliza duas referências mistas: `=$A2*B$1`.

A2 | fx | 1

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		1	2	3	4	5	6	7	8	9	10	11	12
2	1	1	2	3	4	5	6	7	8	9	10	11	12
3	2	2	4	6	8	10	12	14	16	18	20	22	24
4	3	3	6	9	12	15	18	21	24	27	30	33	36
5	4	4	8	12	16	20	24	28	32	36	40	44	48
6	5	5	10	15	20	25	30	35	40	45	50	55	60
7	6	6	12	18	24	30	36	42	48	54	60	66	72
8	7	7	14	21	28	35	42	49	56	63	70	77	84
9	8	8	16	24	32	40	48	56	64	72	80	88	96
10	9	9	18	27	36	45	54	63	72	81	90	99	108
11	10	10	20	30	40	50	60	70	80	90	100	110	120
12	11	11	22	33	44	55	66	77	88	99	110	121	132
13	12	12	24	36	48	60	72	84	96	108	120	132	144
14													

ATENÇÃO

Depois de executar a macro e produzir a tabela de multiplicação na Figura 6.9, observe que o Excel ainda tem o intervalo copiado a partir da linha 2 da macro como o item ativo na área de transferência. Se o usuário dessa macro selecionasse uma célula e pressionasse Enter, o conteúdo dessas células seria copiado para o novo local. Em geral, isso não é desejável. Para tirar o Excel do modo Recortar/Copiar, adicione essa linha de código antes de o programa terminar:

```
Application.CutCopyMode = False
```

Um truque interessante

Tente isto. Mova o ponteiro da célula para F6. Ative a gravação de macro com o botão Gravar Macro na faixa Desenvolvedor. Clique no botão Usar Referências Relativas na faixa Desenvolvedor. Insira a fórmula `=A1` e pressione Ctrl+Enter para permanecer em F6. Clique no botão Parar Gravação na barra de ferramentas flutuante.

Você obtém essa macro de uma única linha, que insere uma fórmula que aponta para uma célula cinco linhas acima e cinco colunas à esquerda:

```
Sub Macro1()
    ActiveCell.FormulaL1C1 = "=L[-5]C[-5]"
End Sub
```

Agora, mova o ponteiro de célula para a célula A1 e execute a macro que você acabou de gravar. Talvez você pense que apontar para uma célula cinco linhas acima de A1 levaria ao onipresente Erro de Tempo de Execução 1004. Mas isso não acontece! Ao executar a macro, a fórmula na célula A1 está apontando para `=XFA1048572`, o que significa que fórmulas no estilo L1C1 realmente recorrem do lado esquerdo da pasta de trabalho para o lado direito. Não consigo pensar em nenhum exemplo no qual isso seja realmente útil, mas para aqueles que contam com a falha do Excel quando se solicita algo que não faz sentido, esteja certo de que a macro fornecerá um resultado e, provavelmente, não o que você esperava!

Lembrando números de coluna associados a letras de coluna

Gosto dessas fórmulas o suficiente para utilizá-las regularmente no VBA, mas não o bastante para mudar minha interface do Excel para números no estilo L1C1. Por isso, tenho de saber rotineiramente que a célula conhecida como U21 é na verdade L21C21.

Saber que U é a 21ª letra do alfabeto não é algo natural. Temos 26 letras, então A é 1 e Z é 26. M é o ponto intermediário do alfabeto, é a Coluna 13. O restante das letras não é muito intuitivo. Descobri que jogando esse pequeno jogo por alguns minutos todos os dias eu havia memorizado rapidamente os números de coluna:

```
Sub QuizColumnNumbers()
    Do
        i = Int(Rnd() * 26) + 1
        Ans = InputBox("Qual número de coluna é a letra " & Chr(64 + i) & "?")
        If Ans = "" Then Exit Do
        If Not (Ans + 0) = i Then
            MsgBox "A letra " & Chr(64 + i) & " é a coluna nº " & i
```

```
        End If
    Loop
End Sub
```

Se você não acha que memorizar números de coluna seja divertido, ou mesmo se algum dia tiver de descobrir o número de coluna da coluna DG, há uma maneira relativamente fácil de fazer isso utilizando a interface do Excel. Mova o ponteiro de célula para a célula A1. Mantenha a tecla Shift pressionada e comece pressionando a tecla de seta para a direita. Para a primeira tela de colunas, o número de coluna aparece na caixa de nome à esquerda da barra de fórmula (veja Figura 6.10).

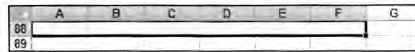
Figura 6.10
Enquanto você seleciona células com o teclado, a caixa Nome exibe quantas colunas são selecionadas para a primeira tela com as colunas.

À medida que você mantém pressionada a tecla de seta para a direita além da primeira tela, uma caixa de dicas de ferramentas à direita da célula atual informa quantas colunas estão selecionadas. Quando chegar à Coluna CS, você será informado de que está na Coluna 97 (veja Figura 6.11).

Figura 6.11
Depois da primeira tela de colunas, uma barra de dicas de ferramentas monitora o número da coluna.

Formatação condicional — L1C1 é um requisito

Ao configurar a formatação condicional, é importante utilizar fórmulas no estilo L1C1. Essa não é uma instrução bem-documentada. O problema é intermitente, mas acho que 1 entre 50 células definidas com a formatação condicional apresentará um comportamento estranho se você utilizar fórmulas no estilo A1. Quando o Excel recebe uma regra de formatação condicional que parece conter uma fórmula no estilo L1C1, supõe que ela esteja utilizando referências no estilo L1C1. O problema é que algumas referências no estilo A1, como R2 para denotar uma única célula, podem ser interpretadas ambiguamente como uma referência no estilo L1C1 para a Linha 2 inteira.

Configurando a formatação condicional na interface com o usuário

A interface com o usuário do Excel destaca a formatação condicional em que o formato da célula se baseia no valor dessa célula. Para a maioria das opções de formatação condicional mostradas na Figura 6.12, o Excel simplesmente tem de examinar o valor na célula atual.

No Excel 2007, a Microsoft introduziu várias opções de formatação condicional em que o formato da célula se baseia na comparação entre a célula e outras células no intervalo selecionado. Por exemplo, Duplicar Valores ou Acima da Média são novos formatos condicionais em que o Excel tem de comparar cada célula com outras em um intervalo. Todas as novas visualizações, como barras de dados, escalas de cor e conjuntos de ícones, comparam as células com outras células no intervalo.

Mas há outro tipo poderoso de formatação condicional em que o formato se baseia em uma fórmula e esta pode apontar para fora do intervalo. No Excel 97-2003, você mudaria a primeira lista suspensa na caixa de diálogo Formatação Condicional de Valor da Célula É para Fórmula É. Muitas pessoas nunca perceberam essa lista suspensa e nunca descobriram a poderosa formatação condicional. No Excel 2007, essa opção também está oculta. É preciso selecionar Formatação Condicional, Realçar Regras das Células, Mais Regras.

Na caixa de diálogo Nova Regra de Formatação, escolha a última opção na lista, Usar uma Fórmula Para Determinar Quais Células Devem Ser Formatadas (veja Figura 6.13). A sintaxe da fórmula permite construir qualquer fórmula ou função que possa ser avaliada como Verdadeira ou Falsa. Essa fórmula pode apontar para células além da célula atual ou combinar várias condições com a função `OU` ou `E`.

Figura 6.12
Tradicionalmente, a formatação condicional define o formato da célula com base apenas no valor em uma única célula.

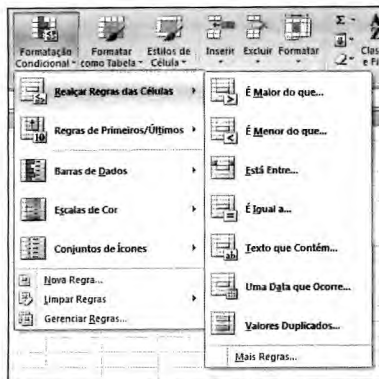
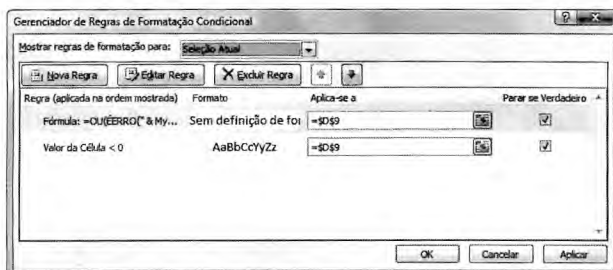



Figura 6.13
Você tem de acessar Mais Regras e então escolher Usar uma Fórmula para alcançar a formatação condicional mais poderosa.

Eis um exemplo relativamente avançado. Queremos ocultar todas as células de erro e pintar os números negativos de vermelho. A idéia básica é definir uma formatação condicional que verifique se a célula contém um erro ou NA. Se qualquer uma for verdadeira, definimos a cor da fonte para corresponder com o segundo plano. Essa primeira condição conta com a sintaxe `Formula Is (Fórmula É)`. A segunda condição é simples e direta e conta com a sintaxe `Cell Value Is (Valor da Célula É)`. A Figura 6.14 mostra como isso seria configurado na interface com o usuário do Excel.

Figura 6.14
Essa célula contém duas formatações condicionais. A primeira foi designada para ocultar qualquer valor de erro e utiliza a poderosa sintaxe `Formula Is`. A segunda condição utiliza a sintaxe mais comum `Value Is` da célula.

Configurando a formatação condicional no VBA

Digamos que você tenha uma pasta de trabalho com uso livre de cores de preenchimento de célula. Você precisa encontrar uma maneira de formatar a fonte como azul-claro quando ocorre um erro em uma célula azul-claro e formatar a fonte como amarelo-claro quando ocorre um erro em uma célula amarela. Uma macro rápida no VBA foi a maneira perfeita para fazer isso.

O objeto `FormatConditions` é utilizado para configurar a formatação condicional. Como cada célula pode ter três `FormatConditions`, o próximo código primeiro exclui todos os formatos condicionais existentes na planilha. Então faz um loop por todas as células que não estiverem em branco na planilha e aplica dois formatos condicionais. No primeiro formato condicional, o tipo é `xlExpression`, o que significa que estamos utilizando a sintaxe `Formula`. Observe que a fórmula especificada para Formula1 está na notação no estilo L1C1. O segundo formato condicional utiliza o tipo `xlCellValue`, que requer que se especifique tanto um operador como um valor. Depois de adicionarmos a condição, configuramos `ColorIndex` para a fonte das condições 1 e 2:

```
Sub ApplySpecialFormattingAll()
    For Each ws In ThisWorkbook.Worksheets
        ws.UsedRange.FormatConditions.Delete
```

```
        For Each cell In ws.UsedRange.Cells
            If Not IsEmpty(cell) Then
                cell.FormatConditions.Add Type:=xlExpression, _
                    Formula1:="=ou(ISERR(LC),isna(LC))"
                cell.FormatConditions(1).Font.Color = cell.Interior.Color
                cell.FormatConditions.Add Type:=xlCellValue, Operator:=xlLess, _
                    Formula1:="0"
                cell.FormatConditions(2).Font.ColorIndex = 3
            End If
        Next cell
    Next ws
End Sub
```

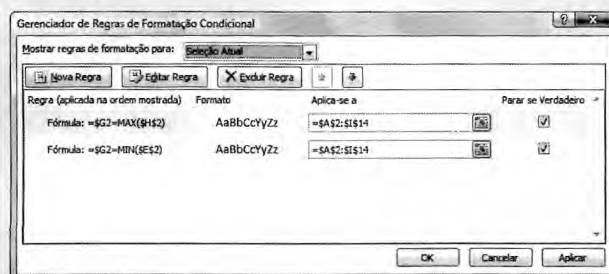
ATENÇÃO

Não utilize fórmulas no estilo A1 para a Formula1 no formato condicional. O código parece funcionar em uma ou algumas células. Entretanto, se aplicar isso a 50 ou mais células, você descobrirá que algumas células em que a fórmula começa apontam para uma célula totalmente diferente. Se utilizar fórmulas no estilo L1C1 para a *Formula1, você não* terá esse problema. Veja o seguinte estudo de caso para obter mais detalhes.

Estudo de caso

Identificando a linha com o maior valor em G

Um dos exemplos clássicos para ilustrar a formatação condicional é esse, que destaca a linha com o valor mínimo e o máximo. A Figura 6.15 mostra como definir isso na interface do Excel.

Figura 6.15 Esse exemplo clássico de formatação condicional indica um problema interessante no código VBA.

Veja o código para criar essa formatação condicional. A fórmula para localizar o valor mais alto na coluna G, expresso no estilo L1C1, é `MAX(C7)`. Isso pode ser resolvido de duas maneiras: com uma fórmula no estilo A1 ou com uma no estilo L1C1. Lembre-se de que a Microsoft começou com o estilo L1C1 e ofereceu o estilo A1 apenas para propósitos de compatibilidade com o Lotus 1-2-3. Isso não está documentado, mas o Excel, claramente, tenta primeiro resolver a fórmula utilizando a formatação no estilo L1C1 e o código a seguir funciona perfeitamente.

```
Sub FindMinMax()
    ' Destaque a linha com a renda mais alta em Verde
    ' Destaque a linha com renda mais baixa em Amarelo
    FinalRow = Cells(Application.Rows.Count, 1).End(xlUp).Row
    With Range("A2:I" & FinalRow)
        .FormatConditions.Delete
        .FormatConditions.Add Type:=xlExpression, Formula1:="=LC7=MAX(C7)"
        .FormatConditions(1).Interior.ColorIndex = 4
        .FormatConditions.Add Type:=xlExpression, Formula1:="=LC7=MIN(C7)"
        .FormatConditions(2).Interior.ColorIndex = 6
    End With
End Sub
```

Agora, imagine o que acontecerá se você tentar configurar um formato condicional apontando para a célula C7 ou R22. É provável que qualquer um desses formatos falhe porque o Excel interpretará a célula C7 como uma referência à Coluna 7.

Com a formatação condicional, o Excel tenta acomodar a fórmula A1 ou L1C1, mas, obviamente, há situações em que a fórmula pode ser interpretada como uma das fórmulas de estilo. Nesse caso, o Excel sempre supõe que o estilo é L1C1, portanto é recomendável utilizar o estilo L1C1 para cada formato condicional que você configurar.

Fórmulas de array requerem fórmulas L1C1

As fórmulas de array são 'superfórmulas' poderosas. No MrExcel.com, refiro-me a essas fórmulas como CSE, porque é preciso utilizar Ctrl+Shift+Enter para inseri-las. Se você não estiver familiarizado com fórmulas de array, parece que elas não funcionam.

A fórmula de array em E20, mostrada na Figura 6.16, faz 18 multiplicações e então soma o resultado. Aparentemente, essa seria uma fórmula inválida e, se você, por acaso, a inserisse sem utilizar Ctrl+Shift+Enter, obteria o esperado erro `#VALOR!` Entretanto, se a inserir com Ctrl+Shift+Enter, a fórmula multiplicará milagrosamente linha por linha e, então, somará o resultado. (Não digite as chaves ao inserir a fórmula.)

Figura 6.16
A fórmula de array em E20 faz 18 multiplicações e depois as soma. Você deve utilizar Ctrl+Shift+Enter para inserir essa fórmula.

E21 | {=SOMA(D$2:D20*E$2:E20)}

	A	B	C	D	E	F	G
1	Região	Produto	Data	Quantidade	Preço unitário	Custo unitário	
2	Leste	XYZ	01/01/2001	1000	22,81	10,22	
3	Centro	DEF	02/01/2001	100	22,57	9,84	
4	Leste	ABC	02/01/2001	500	20,49	8,47	
5	Centro	XYZ	03/01/2001	500	22,48	10,22	
6	Centro	XYZ	04/01/2001	400	23,01	10,22	
7	Leste	DEF	04/01/2001	800	23,19	9,84	
8	Leste	XYZ	04/01/2001	400	22,88	10,22	
9	Centro	ABC	05/01/2001	400	17,15	8,47	
10	Leste	ABC	07/01/2001	400	21,14	8,47	
11	Leste	DEF	07/01/2001	1000	21,73	9,84	
12	Oeste	XYZ	07/01/2001	600	23,01	10,22	
13	Centro	ABC	09/01/2001	800	20,52	8,47	
14	Leste	XYZ	09/01/2001	900	23,35	10,22	
15	Centro	XYZ	10/01/2001	900	23,82	10,22	
16	Leste	XYZ	10/01/2001	900	23,85	10,22	
17	Centro	ABC	12/01/2001	300	20,89	8,47	
18	Oeste	XYZ	12/01/2001	400	22,86	10,22	
19	Centrp	ABC	14/01/2001	100	17,40	8,47	
20	Leste	XYZ	14/01/2001	100	24,01	10,22	
21				Receita Total	234198		
22							

As fórmulas de array em E22:E24 também são poderosas. A fórmula para E22 é:

```
=SOMA(SE(A$2:A$19=$A22,D$2:D$19*E$2:E$19,0))
```

O código para inserir essas fórmulas está a seguir. Embora as fórmulas apareçam na interface com o usuário na notação no estilo A1, você deve utilizar a notação no estilo L1C1 para inserir fórmulas de array:

```
Sub EnterArrayFormulas()
    ' Adiciona uma fórmula para multiplicar preço unitário x quantidade
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    Cells(FinalRow + 1, 5).FormulaArray = "=SOMA(L2C[-1]:L[-1]C[-1]*L2C:L[-1]C)"
End Sub
```

DICA

Para localizar a fórmula L1C1 rapidamente, utilize este truque. Insira uma fórmula no estilo A1 comum ou uma fórmula de array em qualquer célula no Excel. Selecione essa célula, mude para o Editor do VBA e pressione Ctrl+G para exibir a janela Verificação Imediata. Digite `Print ActiveCell.FormulaL1C1` e pressione Enter. O Excel converterá a fórmula na barra de fórmulas em uma fórmula no estilo L1C1.

Próximos passos

A formatação condicional é um dos recursos que foram alterados no Excel 2007 VBA. Leia o Capítulo 7, "O que é novo no Excel 2007 e o que mudou", para aprender vários outros recursos que foram alterados de modo significativo no Excel 2007.

O que é novo no Excel 2007 e o que mudou

Pular

7

Se mudou na interface, mudou no VBA

Felizmente, houve poucas modificações no VBA, mas algumas coisas no modelo de objeto mudaram. Para a maioria dos itens, é óbvio que, como a interface com o usuário do Excel mudou, o VBA também mudou.

> **NOTA** NOVO O ícone Novo é utilizado por todo o livro para indicar as seções que discutem itens que mudaram a partir das versões anteriores. Por exemplo, de modo geral, os nomes (discutidos no Capítulo 8) não mudaram tanto, mas o Gerenciador de Nome mudou; então, há um ícone Novo para chamar sua atenção a isso.

A Faixa

A Faixa é uma das primeiras alterações que você perceberá ao abrir o Excel 2007. Embora o objeto CommandBars ainda funcione até certo ponto, se quiser integrar perfeitamente seus controles personalizados à Faixa, você precisará fazer algumas alterações importantes.

→ **Consulte** o Capítulo 26, "Personalizando a faixa para executar macros", para mais informações.

Gráficos

Gráficos têm muitos recursos novos que não são compatíveis com as versões anteriores do Excel. E embora o programa de gravação de macros grave a maioria das suas ações nas Faixas de opções Design e Layout, ele não gravará a ação na faixa Formato nas caixas de diálogo Formato.

Isso não significa que o código VBA não está disponível; significa apenas que você não consegue aprendê-lo gravando-o.

→ **Consulte** o Capítulo 11, "Criando gráficos", para informações adicionais.

Tabelas dinâmicas

Tabelas dinâmicas têm alguns novos recursos disponíveis que não são compatíveis com as versões anteriores, como subtotais no topo da planilha e as opções de layout de relatório. As tabelas 13.1 e 13.2, no Capítulo 13, "Utilizando o VBA para criar tabelas dinâmicas", relacionam os novos métodos e propriedades no Excel 2007 aos quais você tem de estar atento se precisar tornar uma pasta de trabalho compatível com outras versões.

Formatação Condicional

A formatação condicional foi completamente reformulada. Os locais em que anteriormente estávamos limitados a três condições e a alterar algumas poucas opções de formatação da célula, agora parece que o céu é o limite. Para ter uma idéia de como esse recurso mudou, considere isto: *Special Edition Using*

NESTE CAPÍTULO

Microsoft Office Excel 2007 (ISBN 0-7897-3611-X) tem um capítulo de 40 páginas só para revisar as opções disponíveis. Compare isso com o Excel 2003 e praticamente todos os livros anteriores, em que a cobertura de formatação condicional continha apenas uma ou duas notas de rodapé dispersas pelo livro. Esse recurso é utilizado há muito tempo e, portanto, também o código. Compare as duas macros gravadas a seguir. Ambas são relativamente simples. O preenchimento de uma célula muda para vermelho se o valor na célula estiver entre `1` e `5`. Observe, porém, como há muito mais códigos envolvidos nas novas opções que você agora precisa configurar no 2007.

Macro gravada no Excel 2003:

```
Sub Macro2()
'
'Macro Macro2
'

'
    Selection.FormatConditions.Delete
    Selection.FormatConditions.Add Type:=xlCellValue, Operator:=xlBetween, _
        Formula1:="1", Formula2:="5"
    Selection.FormatConditions(1).Interior.ColorIndex = 3
    ActiveCell.FormulaR1C1 = "2"
    Range("A2").Select
End Sub
```

Macro gravada no Excel 2007:

```
Sub Macro2()
'
'Macro Macro2
'

'
    Selection.FormatConditions.AddColorScale ColorScaleType:=2
    Selection.FormatConditions(Selection.FormatConditions.Count).SetFirstPriority
    Selection.FormatConditions(1).ColorScaleCriteria(1).Type = _
        xlConditionValueNumber
    Selection.FormatConditions(1).ColorScaleCriteria(1).Value = 1
    With Selection.FormatConditions(1).ColorScaleCriteria(1).FormatColor
        .Color = 255
        .TintAndShade = 0
    End With
    Selection.FormatConditions(1).ColorScaleCriteria(2).Type = _
        xlConditionValueNumber
    Selection.FormatConditions(1).ColorScaleCriteria(2).Value = 5
    With Selection.FormatConditions(1).ColorScaleCriteria(2).FormatColor
        .Color = 255
        .TintAndShade = 0
    End With
    ActiveCell.FormulaR1C1 = "2"
    Range("A2").Select
End Sub
```

Tabelas

Tabelas são uma maneira conveniente de lidar com dados que já estão configurados como tabelas (múltiplos registros configurados sob uma linha de cabeçalhos de coluna). Para essa nova funcionalidade, há novos objetos, propriedades e métodos correspondentes.

→ Para aprender mais, **consulte** "Referenciando tabelas," **p. 73** no Capítulo 3, "Referenciando intervalos", e "Tabelas", **p. 150** no Capítulo 8, "Criando e manipula nomes em VBA".

Classificação

Por causa do maior número de opções de classificação (como classificação por cor), o código de classificação passou por algumas alterações. Em vez de uma única linha de código com algumas opções a configurar, você precisa configurar as opções de classificação e então fazer a classificação, como mostrado aqui:

```
Sub Macro2()
'
'Macro Macro2
```

```
    Range("A1:A4").Select
    'Limpa as opções atuais de classificação
    ActiveWorkbook.Worksheets("Sheet1").Sort.SortFields.Clear
    'Configura a nova opção de classificação que é uma simples classificação A-Z
    ActiveWorkbook.Worksheets("Sheet1").Sort.SortFields.Add Key:=Range("A1"), _
        SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:=xlSortNormal
    'Faz a classificação real
    With ActiveWorkbook.Worksheets("Sheet1").Sort
        .SetRange Range("A1:A4")
        .Header = xlYes
        .MatchCase = False
        .Orientation = xlTopToBottom
        .SortMethod = xlPinYin
        .Apply
    End With
End Sub
```

SmartArt

SmartArt é a nova função que substituiu o recurso Diagrama das primeiras versões do Excel. Embora no passado não fosse possível gravar ações de diagrama, você poderia escrever código para elas. No Excel 2007, não é possível gravá-las ou codificá-las. O código do diagrama original também não funciona.

O programa de gravação de macros não gravará ações que ele não gravava nas versões anteriores do Excel

Há uma mudança negativa evidente no VBA. Ações que anteriormente eram graváveis, agora, não são. Por exemplo, no Excel 2003, você podia gravar a inserção de uma caixa de texto e de um texto em uma planilha e obter o seguinte (ligeiramente modificado para caber nessa página):

```
Sub Macro1()
'
'Macro Macro1
'

'
ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal, 268.5, 178.5, _
    116.25, 145.5).Select
Selection.Characters.Text = _
    "This is a test of inserting a text box to a sheet and adding some text." & _
    "This is a test of inserting a text box to a sheet and adding some text." & _
    "This is a test of inserting a text box to a sheet and ad"
Selection.Characters(201).Insert String:="ding some text. "
With Selection.Characters(Start:=1, Length:=216).Font
    .Name = "Arial"
    .FontStyle = "Regular"
    .Size = 10
    .Strikethrough = False
    .Superscript = False
    .Subscript = False
    .OutlineFont = False
    .Shadow = False
    .Underline = xlUnderlineStyleNone
    .ColorIndex = xlAutomatic
End With
Range("I15").Select
End Sub
```

Tente a mesma coisa no Excel 2007 e você obtém isto:

```
Sub Macro1()
'
'Macro Macro1
'
    Selection.Copy
    ActiveSheet.Paste
    ActiveSheet.Paste
    Range("I5").Select
End Sub
```

Isso realmente não é muito útil — o gravador ignora a existência da caixa de texto! Entretanto, não quer dizer que não é possível utilizar o VBA para adicionar uma caixa de texto — simplesmente é um pouco mais difícil descobrir como fazer com que ele faça o que você quer.

Para descobrir como fazer isso funcionar, digitei 'caixa de texto' na Ajuda do Editor do VB e selecionei o método AddTextbox na lista de artigos. A partir daí, modifiquei as partes requeridas do código Excel que havia gravado previamente, o que consistiu em mudar a maneira como a caixa de texto e seu texto são adicionados e, então, deparei com o seguinte:

```
Sub AddTextBox()
ActiveSheet.Shapes.AddTextBox(msoTextOrientationHorizontal, 268.5, 178.5, _
    116.25, 145.5).TextFrame.Characters.Text = _
    "This is a test of inserting a text box to a sheet and adding some text." & _
    "This is a test of inserting a text box to a sheet and adding some text." & _
    "This is a test of inserting a text box to a sheet and adding some text."
With Selection.Characters(Start:=1, Length:=216).Font
    .Name = "Arial"
    .FontStyle = "Regular"
    .Size = 10
    .Strikethrough = False
    .Superscript = False
    .Subscript = False
    .OutlineFont = False
    .Shadow = False
    .Underline = xlUnderlineStyleNone
    .ColorIndex = xlAutomatic
End With
Range("I15").Select
End Sub
```

Aprendendo sobre os novos objetos e métodos

Os arquivos de Ajuda do VBA do Excel contêm várias tabelas relacionadas às alterações nos objetos e métodos no Excel às quais você pode se referir. Elas foram até mesmo divididas pelo número de versão, como mostrado na Figura 7.1. Para acessar essas tabelas, clique no ícone de Ajuda na barra de ferramentas do Editor do VB e selecione 'O que há de novo' na caixa de diálogo que aparece.

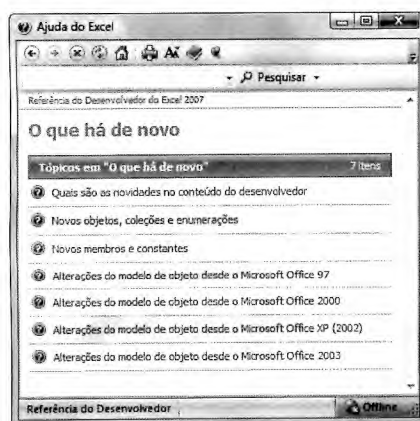
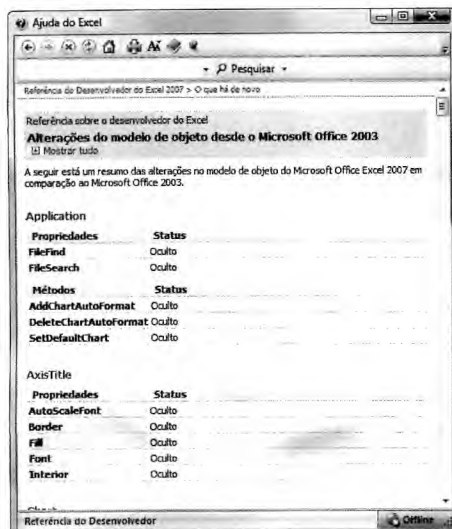
Figura 7.1
A Ajuda do VBA do Excel tem várias seções para ajudá-lo a encontrar o que irá e o que não irá funcionar no novo Excel.

Ao revisar as seções 'Alterações no modelo de objeto desde...', talvez você pense consigo mesmo o que a Microsoft quer dizer quando informa que o status de um item está 'Oculto' (veja Figura 7.2). Se utilizar um desses itens no código, como `FileSearch`, o programa compilará bem, mas não funcionará no Excel 2007. A Microsoft permitiu que você incluísse o item no código para uso legado e ele será compilado, mas quando tentar executá-lo, o programa não saberá o que fazer com ele. A menos que você tenha algum tipo de verificação de modo de compatibilidade no código, seu programa fará a depuração em tempo de execução.

Figura 7.2
Alguns itens aparecem como Oculto nas tabelas de referência Alterações de Modelo de Objeto. Isso significa que o Excel irá compilá-los para uso nas versões legadas, mas eles não irão realmente funcionar no Excel 2007.

Modo de compatibilidade

Com todas as alterações no Excel 2007, agora mais do que nunca, é importante verificar a versão do aplicativo. Duas maneiras de fazer isso são `Version` e `Excel8CompatibilityMode`.

Lidando com questões de compatibilidade

Criar uma pasta de trabalho no modo de compatibilidade pode ser problemático. A maior parte do código ainda executará nas versões anteriores, desde que o programa não encontre um item do modelo de objeto do Excel 2007. Entretanto, se utilizar algum item do modelo de objeto do Excel 2007, o código não compilará nas versões anteriores. Para contornar esse problema, insira um caractere de comentário nas linhas do código específicas à versão 2007, compile e então repita o processo.

Se o único problema em relação ao Excel 2007 for o uso de valores constantes, trate parcialmente seu código como se ele estivesse fazendo uma vinculação tardia com um aplicativo externo (veja a seção "Utilizando valores constantes" no Capítulo 18, "Automatizando o Word", para obter informações adicionais). Se você tiver apenas valores constantes que são incompatíveis, trate-os como argumentos de vinculação tardia, atribuindo a uma variável o valor numérico da constante. A seção a seguir mostra um exemplo disso.

Version

A propriedade `Version` retorna uma string contendo a versão atual do aplicativo Excel. Para a versão 2007, ela é 12. Isso pode se útil se você tiver desenvolvido um suplemento para utilizar em diferentes versões; mas algumas partes dele, como salvar a pasta de trabalho ativa, são específicas à versão:

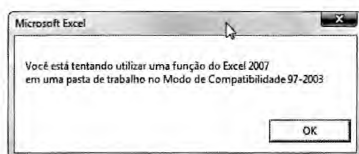
```
Sub wkbkSave()
Dim xlVersion As String
Dim myxlOpenXMLWorkbook As String

myxlOpenXMLWorkbook = "51"

xlVersion = Application.Version

Select Case xlVersion
    Case Is = "9.0", "10.0", "11.0"
        ActiveWorkbook.SaveAs Filename:="LegacyVersionExcel.xls"
    Case Is = "12.0"
        ActiveWorkbook.SaveAs Filename:="Excel2007Version", _
        FileFormat:=myxlOpenXMLWorkbook
End Select
End Sub
```

Observe que para a propriedade `FileFormat`, no caso do Excel 2007, tive de criar minha própria variável `myxlOpenXMLWorkbook` para armazenar o valor constante `xlOpenXMLWorkbook`. Se tentasse executar isso em uma versão anterior do Excel utilizando apenas a constante do Excel 2007, `xlOpenXMLWorkbook`, o código nem mesmo compilaria.

NOVO Excel8CompatibilityMode

Essa propriedade retorna um Booleano, para que você saiba se uma pasta de trabalho está no modo Compatibilidade — isto é, se foi salva como um arquivo do Excel 97-2003. Você utiliza isso, por exemplo, se tiver um suplemento que utiliza a nova formatação condicional, mas não quer que o usuário tente utilizá-lo na pasta de trabalho. A seguinte função, `CompatibilityCheck`, retorna `True` se a pasta de trabalho ativa estiver no modo Compatibilidade, e `False`, se não estiver. O procedimento, `CheckCompatibility`, utiliza o resultado para informar o usuário de que um recurso é incompatível, como mostrado na Figura 7.3:

```
Function CompatibilityCheck() As Boolean
Dim blMode As Boolean

If Application.Version = "12.0" Then
    blMode = ActiveWorkbook.Excel8CompatibilityMode
    If blMode = True Then
        CompatibilityCheck = True
    ElseIf blMode = False Then
        CompatibilityCheck = False
    End If
End If
End Function

Sub CheckCompatibility()
Dim xlCompatible As Boolean

xlCompatible = CompatibilityCheck

If xlCompatible = True Then
    MsgBox "Você está tentando utilizar uma função do Excel 2007 " & Chr(10) & _
    "em uma pasta de trabalho no Modo de Compatibilidade 97-2003"
End If
End Sub
```

Figura 7.3
Utilize `Excel8CompatibilityCheck` para informar um usuário de que certos recursos no seu suplemento não funcionarão em um arquivo do Excel 97-2003 aberto no Excel 2007.

Próximos passos

Agora que tem uma idéia sobre as diferenças com as quais pode se deparar, você está pronto para avançar para o próximo capítulo, que discute como usar intervalos identificados para simplificar a codificação, incluindo um dos meus novos métodos favoritos, o método `Table`.

Crie e manipule nomes no VBA

8

NESTE CAPÍTULO

Nomes no Excel

Você nomeou intervalos em uma planilha realçando um intervalo e digitando um nome na caixa Nome à esquerda do campo de fórmula. Talvez também tenha criado nomes mais complicados que contenham fórmulas — como localizar a última linha em uma coluna. A capacidade de configurar um nome como um intervalo torna muito mais fácil escrever fórmulas e configurar tabelas.

A capacidade de criar e manipular nomes também está disponível no VBA e fornece os mesmos benefícios dos intervalos nomeados em uma planilha: você pode armazenar, por exemplo, um novo intervalo em um nome.

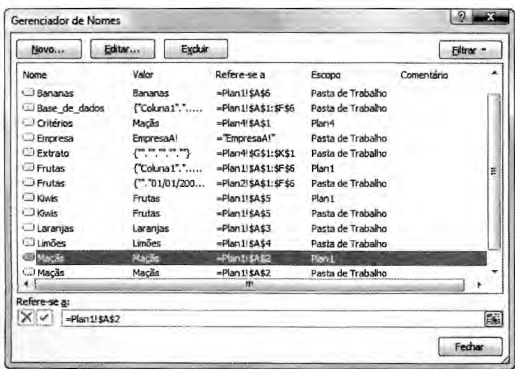
Este capítulo explica os diferentes tipos de nomes e as várias maneiras de utilizá-los.

Nomes globais *versus* nomes locais

Os nomes podem ser *globais* — isto é, disponíveis em qualquer lugar na pasta de trabalho — ou *locais* — disponíveis apenas em uma planilha específica. Com nomes locais, você pode ter diversas referências na pasta de trabalho com o mesmo nome. Nomes globais, porém, devem ser únicos.

NOVO Nas versões anteriores do Excel, era difícil definir entre um nome global e um local — você tinha de estar na planilha correta e comparar a lista de nomes em diferentes planilhas. No Excel 2007, a caixa de diálogo Gerenciador de Nomes relaciona todos os nomes em uma pasta de trabalho, mesmo um nome que tenha sido atribuído tanto no âmbito global como no local. A coluna Escopo relaciona o escopo do nome, seja ele a pasta de trabalho ou uma planilha específica, como Plan1.

Por exemplo, na Figura 8.1, o nome Maçãs é atribuído à Plan1 e também à pasta de trabalho.

Figura 8.1
O Gerenciador de Nomes relaciona todos os nomes locais e globais.

Adicionando nomes

Se gravar a criação de um intervalo nomeado e então visualizar o código, você verá algo parecido com isto:

```
ActiveWorkbook.Names.Add Name:="Frutas", RefersToL1C1:="=Plan2!L1
C1:L6C6"
```

Isso cria um nome global `"Frutas"`, que *inclui* o intervalo A1:F6 (L1C1:R6C6). A fórmula é colocada entre aspas e o sinal de igual deve ser incluído. Além disso, a referência de intervalo deve ser absoluta (inclui o sinal $) ou estar na notação L1C1. Se a planilha em que o nome foi criado for a planilha ativa, a referência de planilha não terá de ser incluída; entretanto, isso pode tornar o código mais fácil de entender.

NOTA

Se a referência não for absoluta, o nome poderá ser criado, mas não apontará para o intervalo correto. Por exemplo, se você executar esta linha do código

```
ActiveWorkbook.Names.Add Name:="Cítricos", _
   RefersToL1C1:="=Plan1!A1"
```

o nome será criado na pasta de trabalho. Como podemos ver na Figura 8.2, porém, na verdade ele não foi atribuído ao intervalo.

Figura 8.2
A célula A1 não recebeu o nome Cítricos porque a fórmula de nomes não tem referências absolutas e não foi reconhecida adequadamente pelo Excel.

Para criar um nome local, inclua o nome da planilha:

```
ActiveWorkbook.Names.Add Name:="Plan2!Frutas", _
    RefersToL1C1:="=Plan2!L1C1:L6C6"
```

Ou especifique que a coleção Nomes pertence a uma planilha:

```
Worksheets("Plan1").Names.Add Name:="Frutas", _
    RefersToL1C1:="=Plan1!L1C1:L6C6"
```

O exemplo anterior é o que você aprenderia no programa de gravação de macros, mas há uma maneira mais simples:

```
Range("A1:F6").Name = "Frutas"
```

Ou, apenas para uma variável local, utilize isto:

```
Range("A1:F6").Name = "Plan1!Frutas"
```

Ao criar nomes com esse método, a referência absoluta não é um requisito.

NOTA

Nomes de tabelas são um novo recurso no Excel 2007. Você pode utilizá-los como nomes definidos, mas eles não são criados da mesma maneira. Veja a seção "Tabelas" mais adiante neste capítulo para informações adicionais sobre como criar nomes de tabelas.

Embora isso seja muito mais fácil e mais rápido que aquilo que o programa de gravação de macros cria, é limitado porque só funciona para intervalos. Fórmulas, strings, números e arrays requerem o uso do método `Add`.

A propriedade `Name` do nome `ObjectName` é um objeto, mas ainda tem uma propriedade `Name`. A linha a seguir renomeia um nome existente:

```
Names("Frutas").Name = "Hortifruti"
```

`Frutas` não existe mais; `Hortifruti` agora é o nome do intervalo.

Ao renomear nomes em que tanto a referência local como a global têm o mesmo nome, a linha anterior renomeia a referência local primeiro.

Se `Range("A1:F6").Name = "Frutas"` existir antes no código e `Range("A1:F6").Name = "Hortifruti"` for adicionado mais tarde, `Hortifruti` sobrescreverá `Frutas`, como mostrado na Figura 8.3. Uma tentativa de acessar o nome local `Frutas` mais tarde no programa cria um erro porque esse nome não existe mais.

Figura 8.3
É fácil sobrescrever um nome existente se você não for cuidadoso. O nome local `Frutas` foi sobrescrito pelo nome local `Hortifrúti`. O nome `Frutas` que você vê é a variável global, não a local.

> **DICA**
> A Validação de Dados (o ícone Validação de Dados na faixa Dados) nos limita a selecionar um intervalo na planilha ativa. Para evitar isso, atribua um nome aos dados que você quer incluir na validação.

Excluindo nomes

Utilize o método `Delete` para excluir um nome:

```
Names("NúmHorti").Delete
```

Ocorrerá um erro se você tentar excluir um nome que não existe.

> **ATENÇÃO**
> Se existirem referências locais e globais com o mesmo nome, seja mais específico quanto ao que estiver sendo excluído.

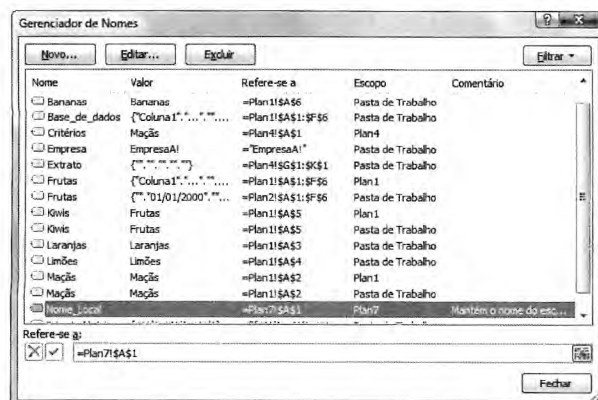
Adicionando comentários

NOVO Com o Excel 2007, você agora pode adicionar comentários a respeito de nomes. Você pode adicionar nome a quaisquer informações adicionais que quiser, por exemplo, por que o nome foi criado ou onde ele é utilizado. Para inserir um comentário para o nome local EscLocal, faça isto:

```
ActiveWorkbook.Worksheets("Plan7").Names("Nome_Local").Comment = _
"Mantém o nome do escritório atual"
```

Os comentários aparecerão em uma coluna no Gerenciador de Nomes, como mostrado na Figura 8.4.

> **ATENÇÃO**
> O nome deve existir antes que um comentário possa ser adicionado a ele.

Figura 8.4
Você pode adicionar comentários a respeito de nomes para ajudá-lo a lembrar-se de qual é o propósito deles.

Tipos de nomes

O uso mais comum dos nomes é para armazenar intervalos, mas eles podem armazenar mais que isso. Afinal, é para isso que eles servem: os nomes armazenam informações. Os nomes tornam simples lembrar onde algo foi gravado, mas utilizam grandes quantidades de informações ou informações potencialmente complexas. E, diferentemente das variáveis, os nomes ajudam a lembrar o que eles armazenam depois da vida do programa.

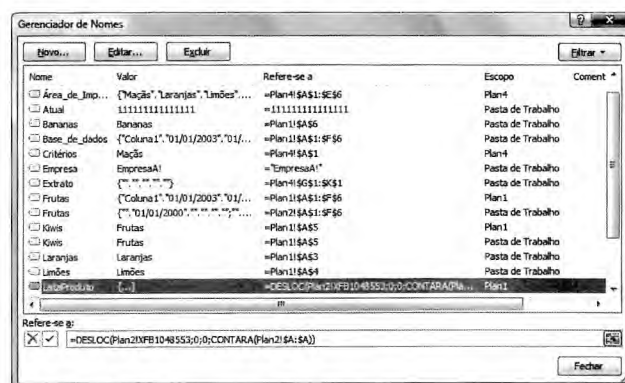
Já abordamos a criação de intervalos nomeados, mas também podemos atribuir nomes a fórmulas de nomes, strings, números e arrays.

Fórmulas

A sintaxe para armazenar uma fórmula em um nome é a mesma de um intervalo, porque o intervalo é, em essência, uma fórmula:

```
Names.Add Name:="ListaProduto", _
    RefersTo:="=DESLOC(Plan2!$A$2;0;0;CONTARA(Plan2!$A:$A))"
```

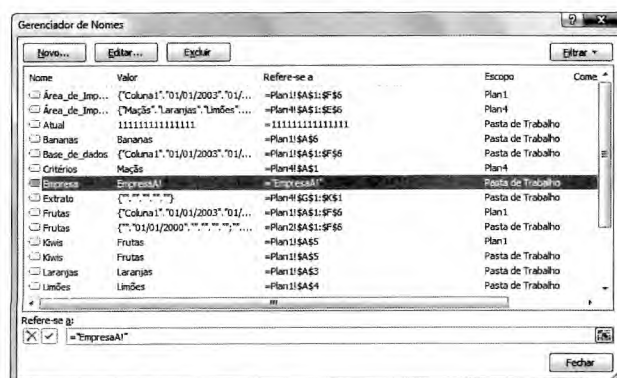
O código anterior leva em consideração uma coluna dinâmica nomeada, o que é muito útil para criar tabelas dinâmicas ou para referenciar qualquer listagem dinâmica em que cálculos podem ser realizados, como mostrado na Figura 8.5.

Figura 8.5
Fórmulas dinâmicas podem ser nomeadas.

Seqüências

Ao utilizar nomes para armazenar strings, como o nome do produtor de frutas atual, coloque o valor da string entre aspas. Como não há nenhuma fórmula envolvida, o sinal de igual não é necessário:

```
Names.Add Name: = "Empresa", RefersTo:="EmpresaA"
```

A Figura 8.6 mostra como o nome codificado apareceria na janela Gerenciador de Nomes.

Figura 8.6
Pode-se atribuir um nome a um valor de string.

Utilizando nomes para armazenar valores

Como os nomes não perdem suas referências entre sessões, essa é uma excelente maneira de armazenar valores em oposição a armazenar valores em células a partir das quais as informações teriam de ser recuperadas. Por exemplo, para monitorar o principal produtor entre as estações, crie um nome como Líder. Se o principal produtor da nova estação corresponder à referência de nome, um relatório especial comparando as estações *poderá ser criado*. A *outra* opção é criar uma planilha especial para monitorar os valores entre sessões e, então, recuperar os valores quando necessário. Com os nomes, os valores estão prontamente disponíveis.

O procedimento a seguir mostra como as células em uma planilha de variáveis são utilizadas para reter informações entre sessões:

```
Sub NoNames(ByRef CurrentTop As String)
PrincipalRevendedor = Worksheets("Variáveis").Range("A1").Value
If CurrentTop = PrincipalRevendedor Then
    MsgBox ("PrincipalProdutor é " & PrincipalRevendedor & " again.")
Else
    MsgBox ("Novo PrincipalProdutor é " & CurrentTop)
End If
End Sub
```

O procedimento a seguir mostra como os nomes são utilizados para armazenar informações entre sessões:

```
Sub WithNames()
If Evaluate("Atual") = Evaluate("Anterior") Then
    MsgBox ("PrincipalProdutor é " & Evaluate("Anterior") & " again.")
Else
    MsgBox ("Novo PrincipalProdutor é " & Evaluate("Atual"))
End If
End Sub
```

Se Atual e Anterior forem nomes declarados anteriormente, você pode acessá-los diretamente em vez de criar variáveis para passá-los. Note o uso do método `Evaluate` para extrair os valores nos nomes. A string armazenada não pode ter mais de 255 caracteres.

Números

Você também pode utilizar nomes para armazenar números entre sessões. Utilize isto:

```
NumDeVendas = 5123
Names.Add Name:="TotalDeVendas", RefersTo:=NumDeVendas
```

Ou isto:

```
Names.Add Name:="TotalDeVendas", RefersTo:=5123
```

Observe a ausência de aspas ou do sinal de igual. O uso de aspas muda o número para uma string; e com a adição de um sinal de igual, o número muda para uma fórmula.

Para recuperar o valor no nome, você tem duas opções:

```
NumDeVendas = Names("TotalDeVendas").Value
```

Ou a versão mais curta:

```
NumDeVendas = [TotalDeVendas]
```

> **DICA** Tenha em mente que alguém que estiver lendo seu código talvez não conheça o uso do método `Evaluate` (colchetes). Se você souber que outra pessoa irá ler seu código, evite o uso do método `Evaluate` ou adicione um comentário explicativo.

Tabelas

NOVO As tabelas do Excel compartilham algumas das propriedades de nomes definidos, mas também têm seus próprios métodos. Diferentemente dos nomes definidos, que são aquilo com o qual estamos acostumados a lidar, as tabelas não podem ser criadas manualmente — isto é, você não pode selecionar um intervalo em uma planilha e digitar um nome no campo Nome, mas pode criá-las *manualmente* usando o VBA.

As tabelas não são criadas utilizando o mesmo método que o dos nomes definidos. Em vez de `Range(xx).Add` ou `Names.Add`, utilizamos `ListObjects.Add`.

Para criar uma tabela a partir das células A1:F6 e, supondo que a tabela tenha cabeçalhos de coluna, como mostrado na Figura 8.7, faça isto:

```
ActiveSheet.ListObjects.Add(xlSrcRange, Range("$A$1:$F$6"), , xlYes).Name = _
"Tabela1"
```

`xlSrcRange` (o `SourceType`) informa ao Excel que a fonte dos dados é um intervalo do Excel. Você então precisa especificar o intervalo (a fonte) da *tabela. Se houver* cabeçalhos na tabela, inclua a linha ao indicar o intervalo. O próximo argumento, não utilizado no

exemplo anterior, é o `LinkSource`, um *Booleano* que indica se há uma fonte de dados externa e se não é utilizada se o `SourceType` for `xlSrcRange`. O `xlYes` faz o Excel saber que a tabela de dados tem cabeçalhos de coluna; se não tiver, o Excel gera-os automaticamente. O argumento final, não mostrado no exemplo anterior, é o destino, utilizado quando o `SourceType` é `xlSrcExternal`, indicando a célula superior esquerda onde a tabela iniciará.

Figura 8.7
Você pode atribuir um nome especial a uma tabela de dados.

Tabela1 | Maçãs

	A	B	C	D	E	F
1	Coluna1	01/01/2003	02/01/2003	03/01/2003	04/01/2003	05/01/2003
2	Maçãs	274	412	159	314	837
3	Laranjas	228	776	344	245	487
4	Limões	160	183	502	583	100
5	Kiwis	478	724	755	618	778
6	Bananas	513	438	600	456	51
7	Total					2253
8						

Utilizando arrays nos nomes

Um nome também pode armazenar os dados armazenados em um array, cujo tamanho é limitado pela memória disponível. Veja o Capítulo 19, "Arrays", para obter mais informações sobre arrays.

Uma referência de array é armazenada em um nome da mesma maneira como uma referência numérica é armazenada:

```
Sub NamedArray()
Dim myArray(10, 5)
Dim i As Integer, j As Integer
'Os loops For a seguir preenchem o array myArray
For i = 1 To 10
    For j = 1 To 5
        myArray(i, j) = i + j
    Next j
Next i
'A linha a seguir seleciona nosso array e atribui um nome
Names.Add Name:="FirstArray", RefersTo:=myArray
End Sub
```

Como o nome está referenciando uma variável, as aspas ou sinais de igual não são necessários.

Nomes reservados

O Excel utiliza nomes locais próprios para monitorar as informações, os quais são considerados reservados e, se você utilizá-los para suas próprias referências, eles poderão causar problemas.

Destaque uma área em uma planilha. Em seguida, na faixa Layout de Página, selecione Área de Impressão, Definir Área de Impressão.

Como mostrado na Figura 8.8, uma listagem Área_de_Impressão está no campo Nome do Intervalo. Remova a seleção da área e examine novamente o campo Nome do Intervalo. O nome ainda está lá. Selecione-o e a área de impressão previamente definida agora permanece destacada. Se você salvar, fechar e reabrir a pasta de trabalho, Área_de_Impressão ainda será configurada como o mesmo intervalo. Área_de_Impressão é um nome reservado pelo Excel para uso próprio.

Figura 8.8
O Excel cria seus próprios nomes.

Área_de_Impressão | Maçãs

	A	B	C	D	E
1	Maçãs	Laranjas	Limões	Kiwis	Bananas
2	274	228	160	478	513
3	412	776	183	724	438
4	159	344	502	755	600
5	314	245	583	618	456
6	837	487	100	778	51
7					

ATENÇÃO

Cada planilha tem uma área de impressão própria. Além disso, configurar uma nova área de impressão em uma planilha com uma área de impressão existente sobrescreve o nome original da área de impressão.

Felizmente, o Excel não tem uma grande lista de nomes reservados:

Critérios

Banco de dados

Extrair

Área_de_Impressão

Títulos_de_Impressão

Critérios e Extrair são utilizados quando o Filtro Avançado (na faixa Dados, selecione Filtro Avançado) é configurado para extrair os resultados do filtro para uma nova localização.

O Banco de Dados não mais é requerido no Excel, mas alguns recursos, como Formulário de Dados, o reconhece. Versões mais antigas do Excel o utilizavam para identificar os dados que se queria manipular em certas funções.

Área_de-Impressão é utilizada quando uma área de impressão é definida (na faixa Layout de Página, selecione Área de Impressão, Definir Área de Impressão) ou quando as opções Configurar Página que designam a área de impressão na faixa Layout de Página, Escala) são alteradas.

Títulos_de_Impressão é utilizado quando títulos de impressão são configurados (Layout de Página, Imprimir Títulos).

Devemos evitar esses nomes e utilizar as variações com cuidado. Por exemplo, se criasse um nome TítulosImpressão, você poderia codificar isto acidentalmente:

```
Worksheets("Plan4").Names("Títulos_de_Impressão").Delete
```

Você acabou de excluir o nome Excel, em vez do seu nome personalizado.

Ocultando nomes

Os nomes são incrivelmente úteis, mas você não necessariamente precisa ver todos aqueles que criou. Como ocorre com muitos outros objetos, nomes têm uma propriedade `Visible`. Para ocultar um nome, configure a propriedade `Visible` como `False`. Para reexibir um nome, configure-a como `True`:

```
Names.Add Name:="NumHortifruti", RefersTo:="=$A$1", Visible:=False
```

> **ATENÇÃO**
>
> Se um usuário criar um objeto Nome com o mesmo nome que aquele oculto, o nome oculto será sobrescrito sem nenhuma mensagem de alerta. Para evitar isso, proteja a planilha.

Verificando a existência de um nome

Você pode utilizar a seguinte função para verificar a existência de um nome definido pelo usuário, até mesmo um oculto, mas isso não retorna a existência dos nomes reservados do Excel. É uma adição útil ao seu arsenal de 'códigos úteis do programador':

```
Function NameExists(FindName As String) As Boolean
Dim Rng As Range
Dim myName As String
On Error Resume Next
myName = ActiveWorkbook.Names(FindName).Name
If Err.Number = 0 Then
    NameExists = True
Else
    NameExists = False
End If
End Function
```

O código anterior também é um exemplo de como utilizar erros a seu favor. Se o nome que você estiver procurando não existir, uma mensagem de erro será gerada. Adicionar a linha `On Error Resume Next` ao início força o código a continuar. Então, você utiliza `Err.Number` para informar se você se deparou com um erro. Se não tiver encontrado um erro, `Err.Number` será zero, o que significa que o nome *existe; caso contrário*, havia um erro e o nome não existia.

Estudo de caso

Utilizando intervalos nomeados por PROCV

Diariamente, você importa um arquivo de dados sobre vendas a partir de uma cadeia de lojas varejistas. O arquivo inclui o número da loja, mas não o nome dela. Obviamente, você não quer digitar os nomes das lojas todos os dias, mas gostaria que eles aparecessem em todos os relatórios que você criar.

Em geral, você inseriria uma tabela dos números e nomes das lojas em uma área que não atrapalhe, na parte de trás de uma planilha. Você pode utilizar o VBA para ajudar a manter diariamente a lista das lojas e, então, utilizar a função PROCV para adicionar os nomes das lojas da lista ao conjunto de dados.

Os passos básicos são:

1. Importar o arquivo de dados.
2. Localizar todos os números de loja únicos no arquivo de hoje.
3. Verificar se alguns desses números de loja não estão na tabela atual dos nomes de loja.
4. Para todas as novas lojas, adicione-as à tabela e pergunte ao usuário o nome de uma loja.
5. A tabela NomeLoja agora é maior, então atribua novamente o intervalo nomeado utilizado para descrever a tabela de lojas.
6. Utilize uma função PROCV (ou, em inglês, VLOOKUP, como no código abaixo) no conjunto de dados original para adicionar o nome de uma loja a todos os registros. A função PROCV referencia o intervalo nomeado da tabela recém-expandida NomeLoja.

O código a seguir trata estes seis passos:

```
Sub ImportData()
'Esta rotina importa vendas.csv para a planilha de dados
'Verifica se alguma loja na coluna A é nova
'Se houver uma nova loja, adiciona-a então à tabela ListaLojas
Dim WSD As Worksheet
Dim WSM As Worksheet
Dim WB As Workbook

Set WB = ThisWorkbook
'Os dados são armazenados na planilha Dados
Set WSD = WB.Worksheets("Dados")
'ListaLojas é armazenada na planilha Menu
Set WSM = WB.Worksheets("Menu")

'Abre o arquivo. Ativa o arquivo csv
Workbooks.Open Filename:="C:\Vendas.csv"
'Copia os dados para WSD e fecha
ActiveWorkbook.Range("A1").CurrentRegion.Copy Destination:=WSD.Range("A1")
ActiveWorkbook.Close SaveChanges:=False

'Localiza uma lista de lojas exclusivas na coluna A
FinalRow = WSD.Cells(WSD.Rows.Count, 1).End(xlUp).Row
WSD.Range("A1").Resize(FinalRow, 1).AdvancedFilter Action:=xlFilterCopy, _
    CopyToRange:=WSD.Range("Z1"), Unique:=True

'Para todas as lojas únicas, verifica se elas estão na
'Lista atual de lojas.
FinalStore = WSD.Range("Z" & WSD.Rows.Count).End(xlUp).Row
WSD.Range("AA1").Value = "There?"
WSD.Range("AA2:AA" & FinalStore).FormulaR1C1 = _
    "=ISNA(VLOOKUP(RC[-1],ListaLojas,1,False))"

'Localiza a próxima linha para uma nova loja. Como ListaLojas inicia em A1
'da planilha Menu, localiza a próxima linha disponível
NextRow = WSM.Range("A" & WSM.Rows.Count).End(xlUp).Row + 1

'Faz um loop pela lista de lojas de hoje. Se elas forem exibidas
'como ausentes, então adicione-as à parte inferior do ListaLojas
For i = 2 To FinalStore
    If WSD.Cells(i, 27).Value = True Then
        ThisStore = Cells(i, 26).Value
        WSM.Cells(NextRow, 1).Value = ThisStore
        WSM.Cells(NextRow, 2).Value = _
```

```
            InputBox(Prompt:="Qual é o nome da loja " _
            & ThisStore, Title:="Nova loja encontrada")
        NextRow = NextRow + 1
    End If
Next i

'Exclui a lista temporária de lojas em Z &AA
WSD.Range("Z1:AA" & FinalStore).Clear

'Se algumas lojas foram adicionadas, redefine o nome ListaLojas
FinalStore = WSM.Range("A" & WSM.Rows.Count).End(xlUp).Row
WSM.Range("A1:B" & FinalStore).Name = "ListaLojas"

'Usa VLOOKUP para adicionar NomeLoja à coluna B do conjunto de dados
WSD.Range("B1").EntireColumn.Insert
WSD.Range("B1").Value = "NomeLoja"
WSD.Range("B2:B" & FinalRow).FormulaL1C1 = "=VLOOKUP(RC1,NomeLoja,2,False)"

'Muda Fórmulas para Valores
WSD.Range("B2:B" & FinalRow).Value = Range("B2:B" & FinalRow).Value

'Libera nossas variáveis
Set WB = Nothing
Set WSD = Nothing
Set WSM = Nothing
End Sub
```

Próximos passos

No próximo capítulo, você aprenderá como o código pode ser escrito para executar automaticamente com base nas ações dos usuários, tais como ativar uma planilha ou selecionar uma célula. Isso é feito com eventos, que são ações no Excel que podem ser capturadas e utilizadas a seu favor.

Programação de eventos

Níveis de eventos

Anteriormente no livro, mencionamos os eventos de pasta de trabalho e apresentamos exemplos de eventos de planilha. Eventos são a maneira como o Excel permite executar código com base em certas ações que acontecem em uma pasta de trabalho.

Esses eventos podem ser encontrados nos seguintes níveis:

- **Nível do aplicativo** — O controle baseado nas ações de aplicativo, como `Application_NewWorkbook`
- **Nível da pasta de trabalho** — O controle baseado em ações da pasta de trabalho, como `Workbook_Open`
- **Nível da planilha** — O controle baseado em ações de planilha, como `Worksheet_SelectionChange`
- **Nível da planilha de gráficos** — O controle baseado em ações de gráfico, como `Chart_Activate`

Os eventos de pasta de trabalho entram no módulo ThisWorkbook (ou EstaPasta_de_trabalho, em português). Os eventos de planilha entram no módulo da planilha que eles afetam (como Sheet1). Os eventos de planilha de gráfico entram no módulo da planilha de gráfico que eles afetam (como Chart1). Os eventos incorporados de gráficos e aplicativo entram nos módulos de classe. Os eventos ainda podem criar chamadas de procedimento ou chamadas de função fora dos próprios módulos. Portanto, se você quiser que a mesma ação aconteça em duas planilhas diferentes, não será necessário copiar o código duas vezes — em vez disso, coloque o código em um módulo e faça cada evento de planilha chamar o procedimento.

Neste capítulo, você aprenderá os diferentes níveis de eventos, onde localizá-los e como utilizá-los.

> **NOTA**
> Eventos de userform e de controle são discutidos no Capítulo 10, "Userforms — uma introdução", e no Capítulo 23, "Técnicas avançadas de userform".

Utilizando eventos

Cada nível consiste em vários tipos de eventos e memorizar a sintaxe de todos eles seria uma proeza. O Excel facilita a visualização e a inserção dos eventos disponíveis diretamente em módulos adequados a partir do Editor do VB.

Quando ThisWorkbook (ou EstaPasta_de_trabalho), Sheet1 (ou Plan1), uma planilha de gráfico ou um módulo de classe estiver ativo, os eventos correspondentes estarão disponíveis por meio das lista suspensas Objeto e Procedimento, como mostrado na Figura 9.1.

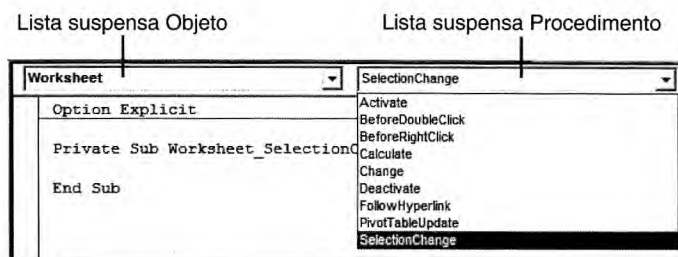
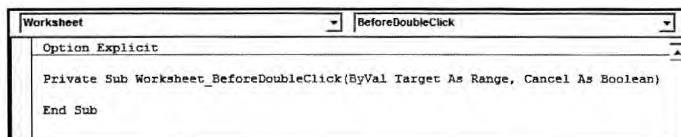
Figura 9.1
Os diferentes eventos são fáceis de serem acessados nas listas suspensas Objeto e Procedimento doEditor do VB.

Depois que o objeto é selecionado, a lista suspensa Procedimento é atualizada e exibe os eventos disponíveis para esse objeto. Selecionar um procedimento posiciona automaticamente o cabeçalho (`Private Sub`) e o rodapé (`End Sub`) do procedimento no editor, como mostrado na Figura 9.2.

Figura 9.2
O cabeçalho e o rodapé do procedimento são posicionados automaticamente.

Parâmetros do evento

Alguns eventos têm parâmetros, como `Target` ou `Cancel`, que permitem que os valores sejam passados para o procedimento. Por exemplo, alguns procedimentos são desencadeados antes do evento real — como `BeforeRightClick`. Atribuir `True` ao parâmetro `Cancel` impede que a ação-padrão aconteça; nesse caso, impede-se que o menu de atalho seja exibido:

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)
Cancel = True
End Sub
```

Ativando eventos

Alguns eventos podem desencadear outros eventos, inclusive eles próprios. Por exemplo, o evento `Worksheet_Change` é desencadeado por uma alteração em uma célula. Se o evento for desencadeado e o próprio procedimento alterar uma célula, o evento será desencadeado novamente, o que mudará uma célula, desencadeará o evento e assim por diante. O procedimento fica preso em um loop sem fim.

Para evitar isso, desative os eventos e depois reative-os no final do procedimento:

```
Private Sub Worksheet_Change(ByVal Target As Range)
Application.EnableEvents = False
Range("A1").Value = Target.Value
Application.EnableEvents = True
End Sub
```

DICA

Para interromper uma macro, pressione Esc ou Ctrl+Break. Para reiniciá-la, utilize Executar na barra de ferramentas ou pressione F5.

Eventos de pasta de trabalho

Os seguintes procedimentos de evento estão disponíveis no nível da pasta de trabalho.

Workbook_Activate()

`Workbook_Activate` ocorre quando a pasta de trabalho contendo esse evento se torna a pasta de trabalho ativa.

Workbook_Deactivate()

`Workbook_Deactivate` ocorre quando a pasta de trabalho ativa é alternada entre a pasta de trabalho contendo o evento e outra pasta de trabalho.

Workbook_Open()

Workbook_Open é o evento de pasta de trabalho padrão. Esse procedimento é ativado quando uma pasta de trabalho é aberta — nenhuma interface com o usuário é necessária. Ele tem uma variedade de usos, como verificar o nome de usuário e personalizar os privilégios do usuário na pasta de trabalho.

O código a seguir verifica o UserName; se não for Admin, esse código protege todas as planilhas contra alterações de usuário. (UserInterfaceOnly permite que macros façam alterações, mas não o usuário.)

```
Private Sub Workbook_Open()
Dim sht As Worksheet
If Application.UserName <> "Admin" Then
    For Each sht In Worksheets
        sht.Protect UserInterfaceOnly:=True
    Next sht
End If
End Sub
```

Você também pode utilizar Workbook_Open para criar menus ou barras de ferramentas personalizados. O código a seguir adiciona o menu MrExcel Programs à Faixa Suplemento com duas opções abaixo dele (veja Figura 9.3).

Figura 9.3
Utilize o evento Open para criar menus personalizados na faixa Suplemento.

➔ Para mais informações sobre como personalizar menus, **consulte** o Capítulo 26, "Personalizando a faixa para executar macros", **p. 543.**

```
Sub Workbook_Open()
Dim cbWSMenuBar As CommandBar
Dim Ctrl As CommandBarControl, muCustom As CommandBarControl
Dim iHelpIndex As Integer
Set cbWSMenuBar = Application.CommandBars("Worksheet menu bar")
iHelpIndex = cbWSMenuBar.Controls("Ajuda").Index
Set muCustom = cbWSMenuBar.Controls.Add(Type:=msoControlPopup, _
    Before:=iHelpIndex, Temporary:=True)
For Each Ctrl In cbWSMenuBar.Controls
    If Ctrl.Caption = "&Programas do MrExcel" Then
        cbWSMenuBar.Controls("Programas do MrExcel").Delete
    End If
Next Ctrl
With muCustom
    .Caption = "&Programas do MrExcel"
    With .Controls.Add(Type:=msoControlButton)
        .Caption = "&Importar e Formatar"
        .OnAction = "ImportFormat"
    End With
    With .Controls.Add(Type:=msoControlButton)
        .Caption = "&Calcular Fim de Ano"
        .OnAction = "CalcYearEnd"
    End With
End With
End Sub
```

Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)

Workbook_BeforeSave ocorre quando a pasta de trabalho é salva. SaveAsUI é configurado como True se for preciso exibir a caixa de diálogo Salvar Como. Cancel configurado como True impede que a pasta de trabalho seja salva.

Workbook_BeforePrint(Cancel As Boolean)

Workbook_BeforePrint ocorre quando qualquer comando de impressão é utilizado — o menu, a barra de ferramentas, o teclado ou a macro. Cancel configurado como True evita que a pasta de trabalho seja impressa.

O código a seguir faz o monitoramento toda vez que uma planilha é impressa. Ele registra em log a data, a hora, o nome de usuário e a planilha impressa em um log de impressão oculto (veja Figura 9.4):

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
```

```
Dim LastRow As Long
Dim PrintLog As Worksheet
Set PrintLog = Worksheets("PrintLog")
LastRow = PrintLog.Cells(PrintLog.Rows.Count, 1).End(xlUp).Row + 1
With PrintLog
    .Cells(LastRow, 1).Value = Now()
    .Cells(LastRow, 2).Value = Application.UserName
    .Cells(LastRow, 3).Value = ActiveSheet.Name
End With
End Sub
```

Figura 9.4
Você pode utilizar o evento `BeforePrint` para manter um log de impressão oculto em uma pasta de trabalho.

	A	B	C
1	Data/Hora	Nome do usuário	Folha Impressa
2	27/01/2007 16:07	Tracy	PrintLog
3			

Você também pode usar o evento `BeforePrint` para adicionar informações a um cabeçalho ou rodapé antes de a planilha ser impressa. Embora agora você possa inserir o caminho de arquivo em um cabeçalho ou rodapé por meio de Configuração de Página, antes do Office XP, a única maneira de adicionar o caminho de arquivo era por meio do código. Este fragmento de código era comumente utilizado:

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    ActiveSheet.PageSetup.RightFooter = ActiveWorkbook.FullName
End Sub
```

Workbook_BeforeClose(Cancel As Boolean)

O `Workbook_BeforeClose` ocorre quando uma pasta de trabalho é fechada. `Cancel` configurado como `True` impede que a pasta de trabalho seja fechada.

Se o evento `Open` for utilizado para criar um menu personalizado, o evento `BeforeClose` será utilizado para excluí-lo:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
Dim cbWSMenuBar As CommandBar
On Error Resume Next
Set cbWSMenuBar = Application.CommandBars("Worksheet menu bar")
cbWSMenuBar.Controls("Programas do MrExcel").Delete
End Sub
```

Esse é um procedimento interessante, mas há um problema: se forem feitas alterações na pasta de trabalho e se ela não for salva, o Excel exibirá a caixa de diálogo Deseja Salvar as Alterações? Essa caixa de diálogo abre depois que o evento `BeforeClose` foi executado. Portanto, se o usuário decidir cancelar, o menu desaparece.

A solução é criar sua própria caixa de diálogo Salvar no evento:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
Dim Msg As String
Dim Response
Dim cbWSMenuBar As CommandBar
If Not ThisWorkbook.Saved Then
    Msg = "Deseja salvar as alterações feitas em" & Me.Name & "?"
    Response = MsgBox(Msg, vbQuestion + vbYesNoCancel)
    Select Case Response
        Case vbYes
            ThisWorkbook.Save
        Case vbNo
            ThisWorkbook.Saved = True
        Case vbCancel
            Cancel = True
            Exit Sub
        End Select
End If
On Error Resume Next
Set cbWSMenuBar = Application.CommandBars("Worksheet menu bar")
cbWSMenuBar.Controls("Programas do MrExcel").Delete
End Sub
```

Workbook_NewSheet(ByVal Sh As Object)

O `Workbook_NewSheet` ocorre quando uma nova planilha é adicionada à pasta de trabalho ativa. `Sh` é o novo objeto Planilha ou Planilha de Gráfico.

Workbook_WindowResize(ByVal Wn As Window)

O `Workbook_WindowResize` ocorre quando a pasta de trabalho ativa é redimensionada. `Wn` é a janela.

> **NOTA**
> Redimensionar apenas a janela ativa da pasta de trabalho inicia esse evento. O redimensionamento da janela de aplicativo é um evento no nível do aplicativo e não é influenciado pelo evento no nível da pasta de trabalho.

Este código desativa o redimensionamento da pasta de trabalho ativa:

```
Private Sub Workbook_WindowResize(ByVal Wn As Window)
Wn.EnableResize = False
End Sub
```

> **ATENÇÃO**
> Se você desativar a capacidade de redimensionamento, os botões minimizar e maximizar serão removidos e a pasta de trabalho não será redimensionada. Para desfazer isso, digite `ActiveWindow.EnableResize = True` na janela Verificação Imediata.

Workbook_WindowActivate(ByVal Wn As Window)

`Workbook_WindowActivate` ocorre quando qualquer janela de pasta de trabalho for ativada. `Wn` é a janela. Esse evento inicia apenas quando a janela de pasta de trabalho é ativada.

Workbook_WindowDeactivate(ByVal Wn As Window)

O `Workbook_WindowDeactivate` ocorre quando qualquer janela da pasta de trabalho é desativada. `Wn` é a janela. Esse evento inicia apenas quando a janela de pasta de trabalho é desativada.

Workbook_AddInInstall()

O `Workbook_AddInInstall` ocorre quando a pasta de trabalho é instalada como um suplemento (selecionando o botão Microsoft Office, Opções do Excel, Suplemento). Dar um clique duplo em um arquivo XLAM (um suplemento) para abri-lo não ativa o evento.

Workbook_AddInUninstall

O `Workbook_AddInUninstall` ocorre quando a pasta de trabalho (suplemento) é desinstalada. O suplemento não é automaticamente fechado.

Workbook_SheetActivate(ByVal Sh As Object)

O `Workbook_SheetActivate` ocorre quando alguma planilha de gráfico ou planilha na pasta de trabalho é ativada. `Sh` é a planilha ativa.

Para afetar uma planilha específica, referencie `Worksheet_Activate`; para planilhas de gráfico, referencie `Chart_Activate`.

Workbook_SheetBeforeDoubleClick (ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)

`Workbook_SheetBeforeDoubleClick` ocorre quando o usuário dá um clique duplo em qualquer planilha de gráfico ou planilha na pasta de trabalho ativa. `Sh` é a planilha ativa; `Target` é o objeto que recebe um clique duplo; `Cancel` configurado como `True` impede que a ação-padrão aconteça.

Para afetar uma planilha específica, referencie `Worksheet_BeforeDoubleClick`; para planilhas de gráfico, referencie `Chart_BeforeDoubleClick`.

Workbook_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)

`Workbook_SheetBeforeRightClick` ocorre quando o usuário clica com o botão direito do mouse em qualquer planilha na pasta de trabalho ativa. `Sh` é a planilha ativa; `Target` é o objeto clicado com o botão direito do mouse; `Cancel` configurado como `True` impede que a ação-padrão aconteça.

Para afetar uma planilha específica, referencie `Worksheet_BeforeRightClick`; para planilhas de gráfico, referencie `Chart_BeforeRightClick`.

Workbook_SheetCalculate(ByVal Sh As Object)

`Workbook_SheetCalculate` ocorre quando qualquer planilha é recalculada ou quaisquer dados atualizados são plotados em um gráfico. `Sh` é a planilha ativa.

Para afetar uma planilha específica, referencie `Worksheet_Calculate`; para planilhas de gráfico, referencie `Chart_Calculate`.

Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Range)

`Workbook_SheetChange` ocorre quando qualquer intervalo em uma planilha é alterado. `Sh` é a planilha; `Target` é o intervalo alterado.

Para afetar uma planilha específica, referencie `Worksheet_Change`.

Workbook_Sync(ByVal SyncEventType As Office.MsoSyncEventType)

`Workbook_Sync` ocorre quando a cópia local de uma planilha em uma pasta de trabalho que faz parte de um Espaço de Trabalho de Documento é sincronizada com a cópia no servidor. `SyncEventType` é o status da sincronização.

Workbook_SheetDeactivate(ByVal Sh As Object)

`Workbook_SheetDeactivate` ocorre quando qualquer planilha de gráfico ou planilha na pasta de trabalho é desativada. `Sh` é a planilha que está sendo modificada.

Para afetar uma planilha específica, referencie `Worksheet_Deactivate`; para planilhas de gráfico, referencie `Chart_Deactivate`.

Workbook_SheetFollowHyperlink (ByVal Sh As Object, ByVal Target As Hyperlink)

`Workbook_SheetFollowHyperlink` ocorre quando qualquer hiperlink é clicado no Excel. `Sh` é a planilha ativa; `Target` é o hiperlink.

Para afetar uma planilha específica, referencie `Worksheet_FollowHyperlink`.

Workbook_SheetSelectionChange(ByVal Sh As Object, ByVal Target As Range)

`Workbook_SheetSelectionChange` ocorre quando um novo intervalo é selecionado em qualquer planilha. `Sh` é a planilha ativa; `Target` é o intervalo afetado.

Para afetar uma planilha específica, referencie `Worksheet_SelectionChange`.

Workbook_PivotTableCloseConnection(ByVal Target As PivotTable)

`Workbook_PivotTableCloseConnection` ocorre quando um relatório de tabela dinâmica fecha sua conexão com a origem dos dados. `Target` é a tabela dinâmica que fechou a conexão.

Workbook_PivotTableOpenConnection(ByVal Target As PivotTable)

`Workbook_PivotTableOpenConnection` ocorre quando um relatório de tabela dinâmica abre uma conexão com a origem dos dados. `Target` é tabela dinâmica que abriu a conexão.

NOVO **Workbook_RowsetComplete**(ByVal Description As String, ByVal Sheet As String, ByVal Success As Boolean)

`Workbook_RowsetComplete` ocorre quando o usuário pesquisa um recordset ou chama a ação rowset em uma Tabela dinâmica OLAP. `Description` é uma descrição do evento; `Sheet` (ou `Plan`) é o nome da planilha em que o recordset é criado; `Success` indica sucesso ou falha.

Eventos de planilha

Os procedimentos de evento a seguir estão disponíveis no nível da planilha.

Worksheet_Activate()

`Worksheet_Activate` ocorre quando a planilha em que está o evento se torna a planilha ativa.

Worksheet_Deactivate()

`Worksheet_Deactivate` ocorre quando outra planilha se torna a planilha ativa.

> NOTA
> Se um evento `Deactivate` estiver na planilha ativa e você mudar para uma planilha com um evento `Activate`, o evento `Deactivate` executará primeiro, seguido pelo evento `Activate`.

Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)

`Worksheet_BeforeDoubleClick` permite controlar o que acontece quando o usuário dá um clique duplo na planilha. `Target` é o intervalo selecionado na planilha; `Cancel` é configurado como `False` por padrão, mas se configurado como `True`, impede que a ação-padrão (como inserir uma célula) aconteça.

O código a seguir impede que o usuário insira uma célula com um clique duplo. E se o campo de fórmula estiver oculto, o usuário não poderá inserir informações da maneira tradicional:

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, _
    Cancel As Boolean)
Cancel = True
End Sub
```

> NOTA
> O código anterior não impede que o usuário dimensione uma linha ou coluna com um clique duplo.

Impedir que o clique duplo insira uma célula permite que ele seja utilizado para outra coisa, como realçar uma célula. O código a seguir muda a cor interna de uma célula para vermelho quando ela recebe um clique duplo:

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, _
    Cancel As Boolean)
Dim myColor As Integer
Target.Interior.ColorIndex = 3
End Sub
```

Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)

`Worksheet_BeforeRightClick` é desencadeado quando o usuário clica com o botão direito do mouse em um intervalo. `Target` é o objeto clicado com o botão direito do mouse; `Cancel` configurado como `True` impede que a ação-padrão aconteça.

Worksheet_Calculate()

`Worksheet_Calculate` ocorre depois que uma planilha é recalculada.

O exemplo a seguir compara os lucros de um mês entre o ano anterior e o ano atual. Se os lucros tiverem caído, aparecerá uma seta vermelha apontando para baixo abaixo do mês; se os lucros tiverem aumentado, aparecerá uma seta verde apontando para cima (veja Figura 9.5):

```
Private Sub Worksheet_Calculate()
Select Case Range("C3").Value
    Case Is < Range("C4").Value
        SetArrow 10, msoShapeDownArrow
    Case Is > Range("C4").Value
        SetArrow 3, msoShapeUpArrow
End Select
End Sub
```

```
Private Sub SetArrow(ByVal ArrowColor As Integer, ByVal ArrowDegree)
'O código a seguir é adicionado a fim de remover as formas anteriores
For Each sh In ActiveSheet.Shapes
    If sh.Name Like "*Arrow*" Then
        sh.Delete
    End If
Next sh
ActiveSheet.Shapes.AddShape(ArrowDegree, 17.25, 43.5, 5, 10).Select
With Selection.ShapeRange
    With .Fill
        .Visible = msoTrue
        .Solid
        .ForeColor.SchemeColor = ArrowColor
        .Transparency = 0#
    End With
    With .Line
        .Weight = 0.75
        .DashStyle = msoLineSolid
        .Style = msoLineSingle
        .Transparency = 0#
        .Visible = msoTrue
        .ForeColor.SchemeColor = 64

.BackColor.RGB = RGB(255, 255, 255)
    End With
End With
Range("A3").Select 'Posiciona a seleção de volta na lista suspensa
End Sub
```

Figura 9.5
Utilize o evento `Calculate` para adicionar elementos gráficos que enfatizam a alteração nos lucros.

	A	B	C
1	Lucro 2005 vs 2006		
2			
3	Junho	**Atual**	3307
4	⇧	**Anterior**	1383
5			
6			
7		**2005**	**2006**
8	Janeiro	3018	7258
9	Fevereiro	9704	3459
10	Março	3950	3874
11	Abril	7518	3907
12	Maio	4542	9774
13	Junho	1383	3307
14	Julho	2888	4741
15	Agosto	8493	8232
16	Setembro	642	9775
17	Outubro	5308	2090
18	Novembro	6040	7490
19	Dezembro	6845	6845
20			

Worksheet_Change(ByVal Target As Range**)**

`Worksheet_Change` é desencadeado por uma alteração no valor de uma célula, por exemplo, quando o texto é inserido, editado ou excluído. `Target` é a célula que foi alterada.

> **NOTA**
> O evento também pode ser desencadeado por meio da colagem de valores. O recálculo de um valor não desencadeia o evento; em vez disso, utilize o evento `Calculation`.

Worksheet_SelectionChange(ByVal Target As Range)

`Worksheet_SelectionChange` ocorre quando um novo intervalo é selecionado. `Target` é o intervalo recentemente selecionado.

O exemplo a seguir ajuda a identificar a célula selecionada destacando a linha e a coluna:

> **ATENÇÃO**
> Esse exemplo utiliza a formatação condicional e sobrescreve toda a formatação condicional existente na planilha. Além disso, o código pode limpar a área de transferência, dificultando copiar e colar na planilha.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
Dim iColor As Integer
On Error Resume Next
iColor = Target.Interior.ColorIndex
If iColor < 0 Then
    iColor = 36
Else
    iColor = iColor + 1
End If
If iColor = Target.Font.ColorIndex Then iColor = iColor + 1
Cells.FormatConditions.Delete
With Range("A" & Target.Row, Target.Address)
    .FormatConditions.Add Type:=2, Formula1:="TRUE"
    .FormatConditions(1).Interior.ColorIndex = iColor
End With
With Range(Target.Offset(1 - Target.Row, 0).Address & ":" & _
    Target.Offset(-1, 0).Address)
    .FormatConditions.Add Type:=2, Formula1:="TRUE"
    .FormatConditions(1).Interior.ColorIndex = iColor
End With
End Sub
```

Worksheet_FollowHyperlink(ByVal Target As Hyperlink)

`Worksheet_FollowHyperlink` ocorre quando um hiperlink recebe um clique. `Target` é o hiperlink.

Estudo de caso

Inserindo rapidamente hora militar em uma célula

Você está inserindo horários de chegada e de partida e quer que os horários sejam formatados com um relógio de 24 horas (hora militar). Você tentou formatar a célula, mas independentemente de como insere as horas, elas são exibidas no formato de horas e minutos, 0:00.

A única maneira de mostrar a hora corretamente, como 23:45, é inseri-la na célula tal como ela está. Mas é demorado digitar os dois-pontos — seria muito mais eficiente inserir apenas os números e deixar o Excel formatar a hora.

A solução? Utilize um evento `Change` para selecionar o que está na célula e inserir o dois-pontos para você:

```
Private Sub Worksheet_Change(ByVal Target As Range)
Dim ThisColumn As Integer
Dim UserInput As String, NewInput As String
ThisColumn = Target.Column
If ThisColumn < 3 Then
    UserInput = Target.Value
    If UserInput > 1 Then
        NewInput = Left(UserInput, Len(UserInput) - 2) & ":" & _
        Right(UserInput, 2)
        Application.EnableEvents = False
        Target = NewInput
        Application.EnableEvents = True
    End If
End If
End Sub
```

Uma entrada de 2345 será exibida como 23:45. Observe que a alteração no formato está limitada às Colunas A e B (`If ThisColumn < 3`) — sem isso, inserir os números em qualquer lugar em uma planilha, como em uma coluna de totais, faria com que ele fosse reformatado.

DICA Utilize `Application.EnableEvents = False` para impedir que o procedimento chame a si mesmo quando o valor no alvo é atualizado.

Eventos de planilha de gráfico

Eventos de gráfico ocorrem quando um gráfico é alterado ou ativado. Os gráficos incorporados exigem o uso de módulos de classe para acessar os eventos.

➔ Para informações adicionais sobre módulos de classe, consulte o Capítulo 22, "Criando classe, registros e coleções", p. 477.

Gráficos incorporados

Como os gráficos incorporados não criam planilhas de gráfico, os eventos de gráfico não estão prontamente disponíveis. Você pode disponibilizá-los adicionando um módulo de classe, como mostrado a seguir:

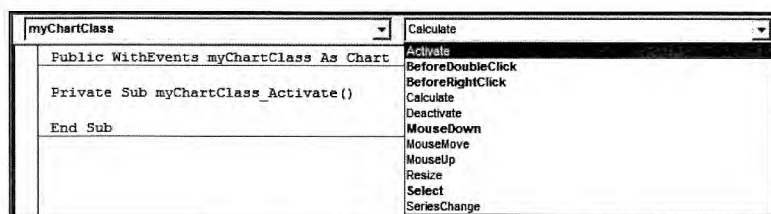
1. Insira um módulo de classe.
2. Renomeie o módulo como `cl_ChartEvents`.
3. Insira a seguinte linha de código no módulo de classe:

   ```
   Public WithEvents myChartClass As Chart
   ```

   Os eventos de gráfico agora estão disponíveis para o gráfico, como mostrado na Figura 9.6. Eles são acessados no módulo de classe em vez de em uma planilha de gráfico.
4. Insira um módulo padrão.
5. Insira as seguintes linhas de código em um módulo padrão:

   ```
   Dim myClassModule As New cl_ChartEvents
   Sub InitializeChart()
       Set myClassModule.myChartClass = _
           Worksheets(1).ChartObjects(1).Chart
   End Sub
   ```

   Essas linhas inicializam o gráfico incorporado a ser reconhecido como um objeto Gráfico. O procedimento deve ser executado uma vez por sessão. (Utilize `Workbook_Open` para automatizar isso.)

Figura 9.6 Os eventos de gráfico incorporado agora estão disponíveis no módulo de classe.

`Chart_Activate()`

`Chart_Activate` ocorre quando uma planilha de gráfico é ativada ou alterada.

`Chart_BeforeDoubleClick(ByVal ElementID As Long, ByVal Arg1 As Long, ByVal Arg2 As Long, Cancel As Boolean)`

`Chart_BeforeDoubleClick` ocorre quando qualquer parte de um gráfico recebe um clique duplo. `ElementID` é a parte do gráfico que recebe um clique duplo, como a legenda; `Arg1` e `Arg2` são dependentes do `ElementID`; `Cancel` configurado como `True` impede que a ação-padrão de um clique duplo ocorra.

O exemplo a seguir oculta a legenda quando ela recebe um clique duplo; dar um clique duplo no eixo abre novamente a legenda:

```
Private Sub MyChartClass_BeforeDoubleClick(ByVal ElementID As Long, _
    ByVal Arg1 As Long, ByVal Arg2 As Long, Cancel As Boolean)
Select Case ElementID
    Case xlLegend
        Me.HasLegend = False
        Cancel = True
    Case xlAxis
        Me.HasLegend = True
        Cancel = True
End Select
End Sub
```

Chart_BeforeRightClick(Cancel As Boolean)

Chart_BeforeRightClick ocorre quando um gráfico é clicado com o botão direito do mouse. Cancel configurado como True impede que a ação- padrão de clicar com o botão direito do mouse ocorra.

Chart_Calculate()

Chart_Calculate ocorre quando os dados de um gráfico são alterados.

Chart_Deactivate()

Chart_Deactivate ocorre quando outra planilha se torna a planilha ativa.

Chart_MouseDown(ByVal Button As Long, ByVal Shift As Long, ByVal x As Long, ByVal y As Long)

Chart_MouseDown ocorre quando o cursor estiver sobre o gráfico e qualquer botão do mouse for pressionado. Button é o botão do mouse que foi clicado; Shift é se uma tecla Shift, Ctrl ou Alt foi pressionada; X é a coordenada X do cursor quando o botão é pressionado; Y é a coordenada Y do cursor quando o botão é pressionado.

O código a seguir é ampliado com um clique do botão esquerdo do mouse e reduzido com um clique do botão direito. Utilize o argumento Cancel no evento BeforeRightClick para tratar os menus que aparecem ao clicar com o botão direito do mouse em um gráfico:

```
Private Sub MyChartClass_MouseDown(ByVal Button As Long, ByVal Shift _
    As Long, ByVal x As Long, ByVal y As Long)
If Button = 1 Then
    ActiveChart.Axes(xlValue).MaximumScale = _
    ActiveChart.Axes(xlValue).MaximumScale - 50
End If
If Button = 2 Then
   ActiveChart.Axes(xlValue).MaximumScale = _
    ActiveChart.Axes(xlValue).MaximumScale + 50
End If
End Sub
```

Chart_MouseMove(ByVal Button As Long, ByVal Shift As Long, ByVal x As Long, ByVal y As Long)

Chart_MouseMove ocorre quando o cursor é movido sobre um gráfico. Button é o botão do mouse que está sendo mantido pressionado, se houver algum; Shift é quando uma tecla Shift, Ctrl ou Alt foi pressionada; X é a coordenada X do cursor no gráfico; Y é a coordenada Y do cursor no gráfico.

Chart_MouseUp(ByVal Button As Long, ByVal Shift As Long, ByVal x As Long, ByVal y As Long)

Chart_MouseUp ocorre quando qualquer botão do mouse é liberado enquanto o cursor está no gráfico. Button é o botão do mouse que foi clicado; Shift é quando a tecla Shift, Ctrl ou Alt foi pressionada; X é a coordenada X do cursor quando o botão é liberado; Y é a coordenada Y do cursor quando o botão é liberado.

Chart_Resize()

Chart_Resize ocorre quando um gráfico é redimensionado por meio das alças de dimensionamento, mas não quando o tamanho é alterado por meio do controle de tamanho na faixa Formato das ferramentas de gráfico.

Chart_Select(ByVal ElementID As Long, ByVal Arg1 As Long, ByVal Arg2 As Long)

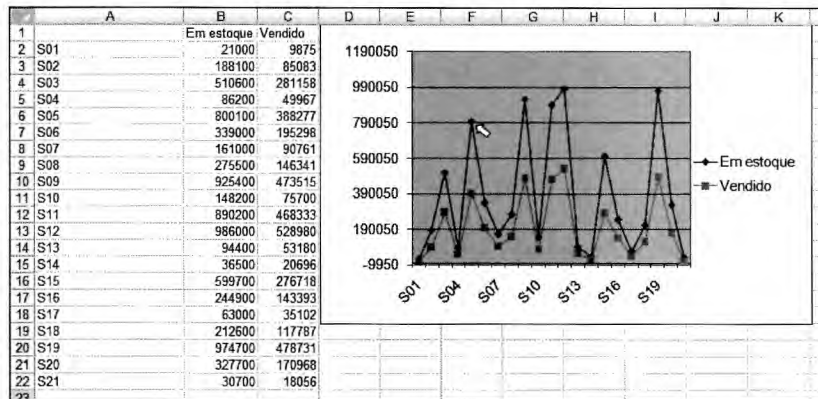
Chart_Select ocorre quando um elemento gráfico é selecionado. ElementID é a parte do gráfico selecionado, como a legenda; Arg1 e Arg2 são dependentes do ElementID.

O código a seguir destaca o conjunto de dados quando um ponto no gráfico é selecionado — supondo que a série inicie em A1 e cada linha seja um ponto a plotar —, como mostrado na Figura 9.7:

```
Private Sub MyChartClass_Select(ByVal ElementID As Long, ByVal Arg1 _
     As Long, ByVal Arg2 As Long)
If Arg1 = 0 Then Exit Sub
```

```
Sheets("Sheet1").Cells.Interior.ColorIndex = xlNone
    If ElementID = 3 Then
     If Arg2 = -1 Then
         ' Selecionada a série inteira em Arg1
         Sheets("Sheet1").Range("A2:A22").Offset(0, Arg1).Interior.ColorIndex = 19
     Else
         ' Selecionado um único ponto no intervalo Arg1, Ponto Arg2
         Sheets("Sheet1").Range("A1").Offset(Arg2, Arg1).Interior.ColorIndex = 19
     End If
 End If
End Sub
```

	A	B	C
1		Em estoque	Vendido
2	S01	21000	9875
3	S02	188100	85083
4	S03	510600	281158
5	S04	86200	49967
6	S05	800100	388277
7	S06	339000	195298
8	S07	161000	90761
9	S08	275500	146341
10	S09	925400	473515
11	S10	148200	75700
12	S11	890200	468333
13	S12	986000	528980
14	S13	94400	53180
15	S14	36500	20696
16	S15	599700	276718
17	S16	244900	143393
18	S17	63000	35102
19	S18	212600	117787
20	S19	974700	478731
21	S20	327700	170968
22	S21	30700	18056

Figura 9.7 Você pode utilizar o evento `Chart_Select` para destacar os dados utilizados para criar um ponto no gráfico.

`Chart_SeriesChange(ByVal SeriesIndex As Long, ByVal PointIndex As Long)`

`Chart_SeriesChange` ocorre quando um ponto no dados do gráfico é atualizado. `SeriesIndex` é o deslocamento na coleção Série da série atualizada; `PointIndex` é o deslocamento na coleção Ponto do ponto atualizado.

`Chart_DragOver()`

`Chart_DragOver` ocorre quando um intervalo é arrastado sobre um gráfico. Esse evento não funciona mais no Excel 2007, mas um programa que o utilize fará uma compilação para uso em versões anteriores.

`Chart_DragPlot()`

`Chart_DragPlot` ocorre quando um intervalo é arrastado e solto em um gráfico. Esse evento não funciona mais no Excel 2007, mas um programa que o utilize fará uma compilação para uso em versões anteriores.

Eventos no nível do aplicativo

Eventos no nível do aplicativo afetam todas as pastas de trabalho abertas em uma sessão do Excel. Esses eventos requerem um módulo de classes para acessá-los (semelhante ao módulo de classe utilizado para acessar eventos para eventos de gráfico incorporado). Siga estes passos para criar o módulo de classe:

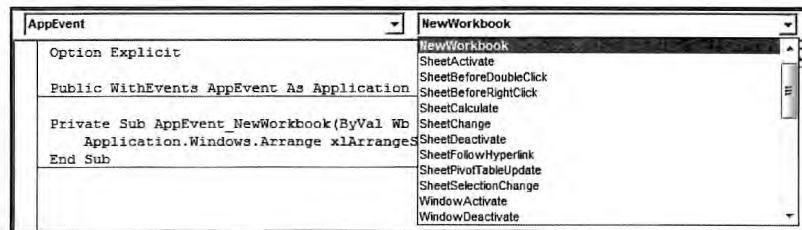
1. Insira um módulo de classe.
2. Renomeie o módulo como `cl_AppEvents`.
3. Insira a seguinte linha de código no módulo de classe:

   ```
   Public WithEvents AppEvent As Application
   ```

   Os eventos de aplicativo estão agora disponíveis para a pasta de trabalho, como mostrado na Figura 9.8. Eles são acessados no módulo de classe em vez de em um módulo-padrão.
4. Insira um módulo-padrão.
5. Insira as seguintes linhas de código no módulo-padrão:

   ```
       Dim myAppEvent As New cl_AppEvents
   Sub InitializeAppEvent()
       Set myAppEvent.AppEvent = Application
   End Sub
   ```

Essas linhas inicializam o aplicativo para que ele reconheça eventos de aplicativo. O procedimento deve ser executado uma vez por sessão (utilize Workbook_Open para automatizar isso).

Figura 9.8
Os eventos de aplicativo estão agora disponíveis por meio do módulo de classe.

NOTA — O objeto na frente do evento, como AppEvent, é dependente do nome atribuído no módulo de classe.

NOVO AppEvent_AfterCalculate()

AppEvent_AfterCalculate ocorre depois que todos os cálculos estão completos e não há nenhuma consulta em andamento ou cálculo incompleto.

NOTA — Esse evento ocorre depois de todos os outros eventos Calculation, AfterRefresh e SheetChange e depois que Application.CalculationState é configurado como xlDone.

AppEvent_NewWorkbook(ByVal Wb As Workbook)

AppEvent_NewWorkbook ocorre quando uma nova pasta de trabalho é criada. Wb é a nova pasta de trabalho. O próximo exemplo organiza as pastas de trabalho abertas em uma configuração lado a lado:

```
Private Sub AppEvent_NewWorkbook(ByVal Wb As Workbook)
    Application.Windows.Arrange xlArrangeStyleTiled
End Sub
```

AppEvent_SheetActivate (ByVal Sh As Object)

AppEvent_SheetActivate ocorre quando uma planilha é ativada. Sh é a planilha (planilha comum ou de gráfico).

AppEvent_SheetBeforeDoubleClick(ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)

AppEvent_SheetBeforeDoubleClick ocorre quando o usuário dá um clique duplo em uma planilha. Target é o intervalo selecionado na planilha; Cancel é configurado como False por padrão, mas, se configurado como True, impede que a ação-padrão (por exemplo, inserir uma célula) aconteça.

AppEvent_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)

AppEvent_SheetBeforeRightClick ocorre quando o usuário clica com o botão direito do mouse em qualquer planilha. Sh é a planilha ativa; Target é o objeto clicado com o botão direito do mouse; Cancel configurado como True impede que a ação-padrão aconteça.

AppEvent_SheetCalculate(ByVal Sh As Object)

AppEvent_SheetCalculate ocorre quando qualquer planilha é recalculada ou quaisquer dados atualizados são plotados em um gráfico. Sh é a planilha ativa.

AppEvent_SheetChange(ByVal Sh As Object, ByVal Target As Range)

AppEvent_SheetChange ocorre quando o valor de qualquer célula é alterado. Sh é a planilha; Target é o intervalo alterado.

AppEvent_SheetDeactivate(ByVal Sh As Object)

AppEvent_SheetDeactivate ocorre quando qualquer planilha de gráfico ou planilha em uma pasta de trabalho é desativada. Sh é a planilha que está sendo desativada.

AppEvent_SheetFollowHyperlink(ByVal Sh As Object, ByVal Target As Hyperlink)

AppEvent_SheetFollowHyperlink ocorre quando qualquer hiperlink é clicado no Excel. Sh é a planilha ativa; Target é o hiperlink.

AppEvent_SheetSelectionChange(ByVal Sh As Object, ByVal Target As Range)

AppEvent_SheetSelectionChange ocorre quando um novo intervalo é selecionado em qualquer planilha. Sh é a planilha ativa; Target é o intervalo selecionado.

AppEvent_WindowActivate(ByVal Wb As Workbook, ByVal Wn As Window)

AppEvent_WindowActivate ocorre quando qualquer janela de pasta de trabalho é ativada. Wb é a pasta de trabalho que está sendo desativada; Wn é a janela.

AppEvent_WindowDeactivate(ByVal Wb As Workbook, ByVal Wn As Window)

AppEvent_WindowDeactivate ocorre quando qualquer janela de pasta de trabalho é desativada. Wb é a pasta de trabalho ativa; Wn é a janela.

AppEvent_WindowResize(ByVal Wb As Workbook, ByVal Wn As Window)

AppEvent_WindowResize ocorre quando a pasta de trabalho ativa é redimensionada. Wb é a pasta de trabalho ativa; Wn é a janela.

ATENÇÃO

Se você desativar a capacidade de redimensionar (EnableResize = False), os botões minimizar e maximizar serão removidos e a pasta de trabalho não poderá ser redimensionada. Para desfazer isso, digite ActiveWindow.EnableResize = True na janela Verificação Imediata.

AppEvent_WorkbookActivate(ByVal Wb As Workbook)

AppEvent_WorkbookActivate ocorre quando qualquer pasta de trabalho é ativada. Wn é a janela. O exemplo a seguir maximiza qualquer pasta de trabalho quando ela é ativada:

```
Private Sub AppEvent_WorkbookActivate(ByVal Wb as Workbook)
    Wb.WindowState = xlMaximized
End Sub
```

AppEvent_WorkbookAddinInstall(ByVal Wb As Workbook)

AppEvent_WorkbookAddinInstall ocorre quando uma pasta de trabalho é instalada como um suplemento (botão Microsoft Office, Opções do Excel, Suplemento). Dar um clique duplo em um arquivo XLAM para abri-lo não ativa o evento. Wb é a pasta de trabalho que está sendo instalada.

AppEvent_WorkbookAddinUninstall(ByVal Wb As Workbook)

AppEvent_WorkbookAddinUninstall ocorre quando uma pasta de trabalho (suplemento) é desinstalada. O suplemento não é fechado automaticamente. Wb é a pasta de trabalho que está sendo desinstalada.

AppEvent_WorkbookBeforeClose(ByVal Wb As Workbook, Cancel As Boolean)

AppEvent_WorkbookBeforeClose ocorre quando uma pasta de trabalho fecha. Wb é a pasta de trabalho; Cancel configurado como True impede a pasta de trabalho de ser fechada.

AppEvent_WorkbookBeforePrint(ByVal Wb As Workbook, Cancel As Boolean)

`AppEvent_WorkbookBeforePrint` ocorre quando qualquer comando Imprimir é utilizado — menu, barra de ferramentas, teclado ou macro. `Wb` é a pasta de trabalho; `Cancel` configurado como `True` impede que a pasta de trabalho seja impressa.

O exemplo a seguir posiciona o nome de usuário no rodapé de cada planilha impressa:

```
Private Sub AppEvent_WorkbookBeforePrint(ByVal Wb As Workbook, _
    Cancel As Boolean)
Wb.ActiveSheet.PageSetup.LeftFooter = Application.UserName
End Sub
```

AppEvent_WorkbookBeforeSave(ByVal Wb As Workbook, ByVal SaveAsUI As Boolean, Cancel As Boolean)

`AppEvent_Workbook_BeforeSave` ocorre quando a pasta de trabalho é salva. `Wb` é a pasta de trabalho; `SaveAsUI` é configurado como `True` se a caixa de diálogo Salvar Como precisar ser exibida; `Cancel` configurado como `True` impede que a pasta de trabalho seja salva.

AppEvent_WorkbookNewSheet(ByVal Wb As Workbook, ByVal Sh As Object)

`AppEvent_WorkbookNewSheet` ocorre quando uma nova planilha é adicionada à pasta de trabalho ativa. `Wb` é a pasta de trabalho; `Sh` é a nova planilha ou objeto de planilha de gráfico.

AppEvent_WorkbookOpen(ByVal Wb As Workbook)

`AppEvent_WorkbookOpen` ocorre quando uma pasta de trabalho é aberta. `Wb` é a pasta de trabalho que acabou de ser aberta.

AppEvent_WorkbookPivotTableCloseConnection(ByVal Wb As Workbook, ByVal Target As PivotTable)

`AppEvent_PivotTableCloseConnection` ocorre quando um relatório de tabela dinâmica fecha sua conexão com sua origem de dados. `Wb` é a pasta de trabalho contendo a tabela dinâmica que desencadeou o evento; `Target` é tabela dinâmica que fechou a conexão.

AppEvent_WorkbookPivotTableOpenConnection(ByVal Wb As Workbook, ByVal Target As PivotTable)

`AppEvent_PivotTableOpenConnection` ocorre quando um relatório de tabela dinâmica abre uma conexão com sua origem de dados. `Wb` é a pasta de trabalho contendo a tabela dinâmica que desencadeou o evento; `Target` é a tabela dinâmica que abriu a conexão.

AppEvent_WorkbookRowsetComplete(ByVal Wb As Workbook, ByVal Description As String, ByVal Sheet As String, ByVal Success As Boolean)

`AppEvent_RowsetComplete` ocorre quando o usuário pesquisa um recordset ou chama à ação rowset em uma tabela dinâmica OLAP. `Wb` é a pasta de trabalho que desencadeou o evento; `Description` é uma descrição do evento; `Sheet` é o nome da planilha em que o recordset é criado; `Success` indica sucesso ou falha.

AppEvent_WorkbookSync(ByVal Wb As Workbook, ByVal SyncEventType As Office.MsoSyncEventType)

`AppEvent_Workbook_Sync` ocorre quando a cópia local de uma planilha em uma pasta de trabalho que faz parte de um Espaço de Trabalho de Documento é sincronizada com a cópia no servidor. `Wb` é a pasta de trabalho que desencadeou o evento; `SyncEventType` é o status da sincronização.

Próximos passos

Neste capítulo, você aprendeu mais sobre como interfacear com o Excel. O próximo capítulo apresenta as ferramentas que podem ser usadas para interagir com os usuários, solicitar-lhes informações para uso no código, alertá-los sobre ações inválidas ou simplesmente fornecer-lhes uma interface com a qual possam trabalhar além da planilha.

Userforms — uma introdução

10

NESTE CAPÍTULO

Métodos de interação com o usuário

Com os userforms, você exibe informações e permite que o usuário insira informações. Os controles `InputBox` e `MsgBox` são maneiras simples de fazer isso. Você pode usar os controles de userform no Editor do VB para criar formulários mais complexos.

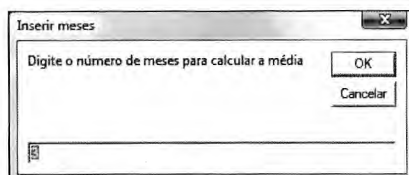
Este capítulo abrange uma interface com o usuário simples, que usa caixas de entrada e caixas de mensagem e os princípios básicos da criação de userforms no Editor do VB. Para a programação mais avançada, veja o Capítulo 23, "Técnicas avançadas de userform".

Caixas de entrada

A função `InputBox` é utilizada para criar um elemento básico de interface que solicita a entrada do usuário antes de o programa avançar. Você pode configurar o prompt, o título da janela, um valor padrão, a posição da janela e arquivos de ajuda de usuário. Apenas dois botões são fornecidos: OK e Cancelar. O valor retornado é uma string.

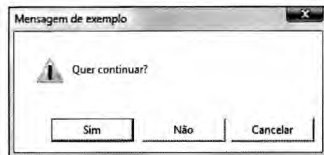
O seguinte código solicita ao usuário o número de meses em que calcular a média. A Figura 10.1 mostra a `InputBox` resultante.

```
AveMos = InputBox(Prompt:="Digite o número " & _
" de meses para calcular a média", Title:="Inserir meses", _
Default:="3")
```

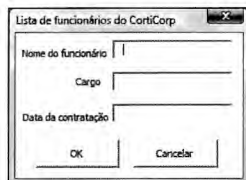
Figura 10.1
Uma simples, mas efetiva, caixa de entrada.

Caixas de mensagem

A função `MsgBox` cria uma caixa de mensagem que exibe informações e espera o usuário clicar em um botão antes de continuar. Enquanto `InputBox` tem apenas o botões OK e Cancelar, `MsgBox` oferece várias configurações de botões, que incluem Sim, Não, OK e Cancelar. Você também pode configurar o prompt, o título da janela e os arquivos de ajuda. O código seguinte produz um prompt simples para descobrir se o usuário quer continuar. Uma instrução `Select Case` é utilizada no programa com a ação apropriada. A Figura 10.2 mostra a caixa de mensagem personalizada resultante.

```
MyMsg = "Quer continuar?"
Response = MsgBox(myMsg, vbExclamation + vbYesNoCancel, myTitle)
Select Case Response
    Case Is = vbYes
        ActiveWorkbook.Close SaveChanges:=False
    Case Is = vbNo
        ActiveWorkbook.Close SaveChanges:=True
    Case Is = vbCancel
        Exit Sub
End Select
```

Figura 10.2
A função `MsgBox` é utilizada para exibir informações e obter uma resposta básica do usuário.

Criando um userform

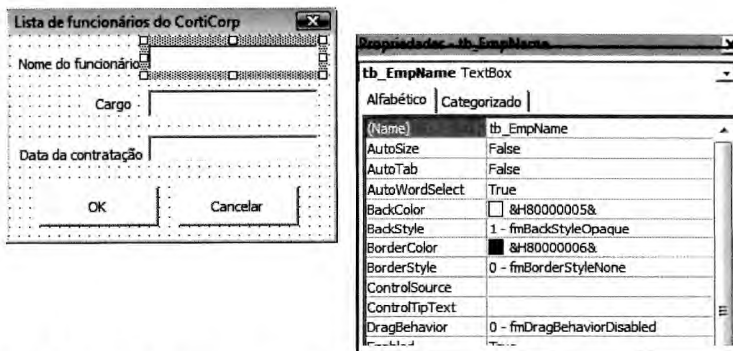
Os userforms combinam as capacidades de `InputBox` e `MsgBox` para criar uma maneira mais eficiente de interagir com o usuário. Por exemplo, em vez de fazer o usuário preencher as informações pessoais em uma planilha, você pode criar um userform que solicita os dados necessários (veja Figura 10.3).

Figura 10.3
Você pode criar um userform personalizado para obter informações adicionais do usuário.

Insira um userform no Editor do VB escolhendo Inserir, UserForm, no menu principal. Um módulo UserForm é adicionado ao Project Explorer, um formulário em branco aparece na janela em que o código normalmente está e a caixa de ferramentas Controles aparece.

Você pode redimensionar o formulário pegando e arrastando as alças no lado direito, parte inferior ou canto inferior direito do userform. Para adicionar controles ao formulário, clique no controle desejado na caixa de ferramentas arraste-o até o formulário. Os controles podem ser movidos e redimensionados a qualquer hora.

> **NOTA** A caixa de ferramentas, por padrão, exibe os controles mais comuns. Para acessar mais controles, clique com o botão direito do mouse na caixa de ferramentas e selecione Controles Adicionais. Mas tenha cuidado; outros usuários talvez não tenham os mesmos controles adicionais que você tem. Se você enviar um formulário com um controle que eles não instalaram, o programa gerará um erro.

Depois que um controle é adicionado a um formulário, suas propriedades podem ser alteradas na janela Propriedades. Essas propriedades podem ser configuradas manualmente, agora ou mais tarde, de modo programático. Se a janela Propriedades não estiver visível, você pode abri-la selecionando Exibir, Janela 'Propriedades'. A Figura 10.4 mostra a janela Propriedades para uma caixa de texto.

Figura 10.4
Utilize a janela Propriedades para alterar as propriedades de um controle.

Chamando e ocultando um userform

Um userform pode ser chamado a partir de qualquer módulo. `FormName.Show` abre um formulário para o usuário:

```
frm_AddEmp.Show
```

O método `Load` também pode ser utilizado para chamar um userform. Isso permite que um formulário seja carregado, mas permaneça oculto:

```
Load frm_AddEmp
```

Para ocultar um userform, utilize o método `Hide`. O formulário continuará ativo, mas agora não poderá ser visto pelo usuário. Os controles no formulário ainda poderão ser acessados programaticamente:

```
Frm_AddEmp.Hide
```

O método `Unload` descarrega o formulário da memória e o remove da visualização do usuário. O formulário não poderá mais ser acessado pelo usuário ou de maneira programática:

```
Unload Me
```

Programando o userform

O código de um controle entra no módulo de formulários. Diferentemente dos outros módulos, dar um clique duplo no módulo Formulários abre o formulário no Modo de Criação. Para visualizar o código, clique com o botão direito do mouse no módulo ou no userform no Modo de criação e selecione Exibir Código.

Eventos de userform

Exatamente como uma planilha, um userform tem eventos desencadeados por ações. Depois que o userform foi adicionado ao projeto, os eventos estarão disponíveis na lista suspensa Propriedades, no canto superior direito da janela Código (veja Figura 10.5), selecionando-se UserForm na lista suspensa Objeto, à esquerda.

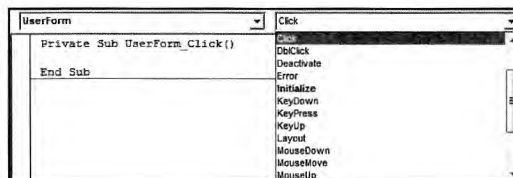
Figura 10.5
Vários eventos para o userform podem ser selecionados na lista suspensa na parte superior da janela Código.

Os eventos disponíveis para userforms estão descritos na Tabela 10.1.

Tabela 10.1 Os eventos para Userforms

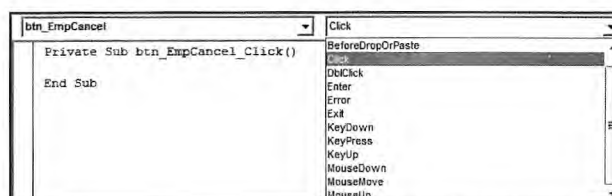
Evento	Descrição
`Activate`	Ocorre quando um userform é mostrado ao ser carregado ou reexibido. Esse evento é desencadeado depois do evento `Initialize`.
`AddControl`	Ocorre quando um controle é adicionado a um userform em tempo de execução. Não executa em tempo de projeto ou na inicialização de userform.
`BeforeDragOver`	Ocorre enquanto o usuário faz uma operação de arrastar e soltar sobre o userform.
`BeforeDroporPaste`	Ocorre antes de o usuário estar prestes a soltar ou colar dados no userform.
`Click`	Ocorre quando o usuário clica no userform com o mouse.
`DblClick`	Ocorre quando o usuário dá um clique duplo no userform com o mouse.
`Deactivate`	Ocorre quando um userform é desativado.
`Error`	Ocorre quando o userform depara com um erro e não pode retornar informações sobre o erro.
Evento	Descrição
`Initialize`	Ocorre quando o userform é carregado pela primeira vez, antes do evento `Activate`. Se você ocultar e depois exibir um formulário, `Initialize` não será desencadeado.
`KeyDown`	Ocorre quando o usuário pressiona uma tecla no teclado.
`KeyPress`	Ocorre quando o usuário pressiona uma tecla ANSI. Uma tecla ANSI é um caractere digitável, como a letra *A*. Um caractere não-digitável seria, por exemplo, a tecla Tab.
`KeyUp`	Ocorre quando o usuário solta uma tecla no teclado.

Tabela 10.1 Os eventos para Userforms *(cont.)*

Evento	Descrição
`Layout`	Ocorre quando o controle muda de tamanho.
`MouseDown`	Ocorre quando o usuário pressiona o botão do mouse dentro das bordas do userform.
`MouseMove`	Ocorre quando o usuário move o mouse dentro das bordas do userform.
`MouseUp`	Ocorre quando o usuário libera o botão do mouse dentro das bordas do userform.
`QueryClose`	Ocorre antes de um userform fechar. Permite reconhecer o método utilizado para fechar um formulário e faz o código responder de maneira correspondente.
`RemoveControl`	Ocorre quando um controle é excluído do userform.
`Resize`	Ocorre quando o userform é redimensionado.
`Scroll`	Ocorre quando a caixa de barra de rolagem, se visível, é reposicionada.
`Terminate`	Ocorre depois que o userform foi descarregado. É desencadeado depois de `QueryClose`.
`Zoom`	Ocorre quando o valor de zoom é alterado.

Programando controles

Para programar um controle, destaque-o e selecione Exibir, Código. O rodapé, o cabeçalho e a ação-padrão para o controle são automaticamente inseridos no campo de programação. Para ver as outras ações disponíveis para um controle, selecione o controle na lista suspensa Objeto e visualize as ações na lista suspensa Propriedades, como mostrado na Figura 10.6.

Figura 10.6
Várias ações para um controle podem ser selecionadas nas listas suspensas do Editor do VB.

Os controles são objetos, como `ActiveWorkbook`. Eles têm propriedades e métodos, dependentes do tipo de controle. A maior parte da programação para os controles é feita atrás do formulário; mas se outro módulo precisar referir-se a um controle, o pai, que é o formulário, terá de ser incluído com o objeto.

```
Private Sub btn_EmpCancel_Click()
Unload Me
End Sub
```

O código anterior pode ser dividido em três seções:

- `btn_EmpCancel` — Nome atribuído ao controle
- `Click` — Ação do controle
- `Unload Me` — O código atrás do controle (nesse caso, descarregar o formulário)

Estudo de caso

Adicionando controles a um formulário existente

Se você tiver um userform utilizado há algum tempo e mais tarde tentar adicionar um novo controle, você poderá achar que o Excel fica confuso com o controle. Verá que o controle é adicionado ao formulário, mas quando você clica com o botão direito do mouse nele e escolhe Exibir, Código, o módulo de código não parece reconhecer que o controle existe. O nome de controle não estará disponível na lista suspensa à esquerda, na parte superior do módulo de código.

Para contornar essa situação, siga estes passos:

1. Adicione todos os controles que você precisa acrescentar ao formulário existente.
2. No Project Explorer, clique com o botão direito do mouse no userform e escolha Exportar Arquivo. Escolha Salvar *para salvar o arquivo* na localização-padrão.
3. *No Project* Explorer, clique com o botão direito do mouse no userform e escolha Remove. Como você acabou de exportar, pode clicar em Não para a pergunta relacionada à exportação.
4. Clique com o botão direito do mouse em qualquer lugar no Project Explorer e escolha Importar Arquivo. Selecione o nome de arquivo que você salvou no passo 2.

Os novos controles agora estarão disponíveis no painel de código do userform.

Utilizando controles de formulários básicos

Todos os controles têm diferentes eventos associados, o permite que você codifique o que acontece com base nas ações do usuário. Uma tabela que revisa os eventos de controle está disponível no final de cada uma das seções a seguir.

Utilizando rótulos, caixas de texto e botões de comando

Nosso formulário básico, como mostrado na Figura 10.7, consiste em botões de rótulo, caixas de texto e botões de comando. Ele é um método simples, mas eficaz de solicitar informações do usuário.

Depois que as caixas de texto foram preenchidas, o usuário clica em OK e as informações são adicionadas a uma planilha (veja Figura 10.8).

Figura 10.7
Formulário simples para coletar informações do usuário.

	A	B	C
1	Adicionar	Exibir	
2	Nome	Cargo	Data Contratação
3	Tracy Syrstad	Project Consultant	20-jul-03
4	Cort Childon-Hoff	CEO	1-jan-01
5			

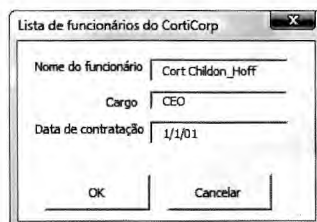



Figura 10.8
As informações são adicionadas à planilha

```
Private Sub btn_EmpOK_Click()
Dim LastRow As Long
LastRow = Worksheets("Funcionario").Cells(Worksheets("Funcionario").Rows.Count, 1) _
.End(xlUp).Row + 1
Cells(LastRow, 1).Value = tb_EmpName.Value
Cells(LastRow, 2).Value = tb_EmpPosition.Value
Cells(LastRow, 3).Value = tb_EmpHireDate.Value
End Sub
```

Com uma alteração no código, como mostrado no exemplo a seguir, o mesmo design de formulário pode ser utilizado para recuperar informações. O exemplo de código a seguir recupera a posição e a data de contratação *depois que o nome* do funcionário foi inserido:

```
Private Sub btn_EmpOK_Click()
Dim EmpFound As Range
With Range("EmpList") 'um intervalo nomeado em uma planilha listando os nomes de funcionários
    Set EmpFound = .Find(tb_EmpName.Value)
    If EmpFound Is Nothing Then
        MsgBox ("Funcionário não localizado!")
        tb_EmpName.Value = ""
        Exit Sub
    Else
```

```
        With Range(EmpFound.Address)
            tb_EmpPosition = .Offset(0, 1)
            tb_HireDate = .Offset(0, 2)
        End With
    End If
End With
End Sub
```

Os eventos disponíveis para os controles `Label`, `TextBox` e `CommandButton` são descritos na Tabela 10.2.

Tabela 10.2 Os eventos para os controles `Label`, `TextBox` e `CommandButton`

Evento	Descrição
`AfterUpdate`[2]	Ocorre depois que os dados do controle foram alterados pelo usuário.
`BeforeDragOver`	Ocorre enquanto o usuário arrasta e solta dados sobre o controle.
`BeforeDropOrPaste`	Ocorre antes de o usuário estar prestes a soltar ou colar dados no controle.
`BeforeUpdate`[2]	Ocorre antes dos dados no controle ser alterados.
`Change`[2]	Ocorre quando o valor do controle é alterado.
`Click`[1,3]	Ocorre quando o usuário clica no controle com o mouse.
`DblClick`	Ocorre quando o usuário dá um clique duplo no controle com o mouse.
`DropButtonClick`[2]	Ocorre quando o usuário pressiona F4 no teclado. Isso é semelhante ao controle suspenso na caixa de combinação, mas não há nenhuma lista suspensa em uma caixa de texto.
`Enter`[2,3]	Ocorre antes de o controle receber o foco de outro controle no mesmo userform.
`Error`	Ocorre quando o controle encontra um erro e não consegue retornar as informações sobre o erro.
`Exit`[2,3]	Ocorre assim que o foco do controle passa para outro controle no mesmo userform.
`KeyDown`[2,3]	Ocorre quando o usuário pressiona uma tecla no teclado.
`KeyPress`[2,3]	Ocorre quando o usuário pressiona uma tecla ANSI. Uma tecla ANSI é um caractere digitável, como a letra *A*. Um caractere não-digitável seria, por exemplo, a tecla Tab.
`KeyUp`[2,3]	Ocorre quando o usuário solta uma tecla no teclado.
`MouseDown`	Ocorre quando o usuário pressiona o botão do mouse dentro das bordas do controle.
`MouseMove`	Ocorre quando o usuário move o mouse dentro das bordas do controle.
`MouseUp`	Ocorre quando o usuário solta o botão do mouse dentro das bordas do controle.

[1] Controle *`Label`* somente

[2] Controle *`TextBox`* somente

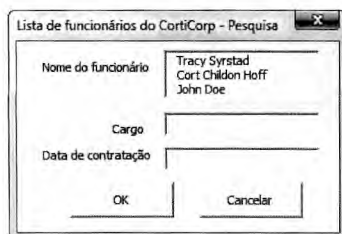
[3] Controle *`CommandButton`* somente

Decidindo entre caixas de listagem ou caixas de combinação para uso em formulários

Você pode permitir que os usuários localizem o nome de um funcionário digitando-o; mas, e se eles digitarem o nome incorretamente? Você terá de encontrar uma maneira de confirmar que o nome foi inserido corretamente. O que você utiliza: uma caixa de listagem ou uma caixa de combinação?

- Uma caixa de listagem exibe uma lista de valores entre os quais o usuário pode escolher.
- Uma caixa de combinação exibe uma lista de valores entre os quais o usuário pode escolher e permite que o usuário insira um novo valor.

Nesse caso, como queremos limitar as opções do usuário, utilizamos uma caixa de listagem para relacionar os nomes dos funcionários, como mostrado na Figura 10.9.

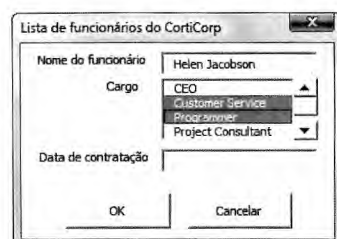
Figura 10.9
Utilize uma caixa de listagem para controlar a entrada de usuário.

Na propriedade `RowSource` da caixa da listagem, insira o intervalo a partir do qual o controle deve selecionar os dados. Utilize um intervalo nomeado dinâmico para manter a lista atualizada se mais funcionários forem adicionados:

```
Private Sub btn_EmpOK_Click()
Dim EmpFound As Range
With Range("EmpList")
    Set EmpFound = .Find(lb_EmpName.Value)
    If EmpFound Is Nothing Then
        MsgBox ("Funcionário não localizado!")
        lb_EmpName.Value = ""
        Exit Sub
    Else
        With Range(EmpFound.Address)
            tb_EmpPosition = .Offset(0, 1)
            tb_HireDate = .Offset(0, 2)
        End With
    End If
End With
End Sub
```

Utilizando a propriedade `MultiSelect` de uma caixa de listagem

As caixas de listagem têm uma propriedade `MultiSelect`, que permite ao usuário selecionar múltiplos itens a partir das escolhas na caixa de listagem, como mostrado na Figura 10.10:

- `fmMultiSelectSingle` — A configuração-padrão permite que seja selecionado um único item por vez.
- `fmMultiSelectMulti` — Permite que se remova a seleção de um item clicando-se nele novamente; múltiplos itens também podem ser selecionados.
- `fmMultiSelectExtended` — Permite que as teclas Ctrl e Shift sejam utilizadas para selecionar múltiplos itens.

Figura 10.10
`MultiSelect` pode permitir ao usuário selecionar múltiplos itens em uma caixa de listagem.

Se vários itens são selecionados, a propriedade `Value` não pode ser usada para recuperá-los. Em vez disso, verifique se o item está selecionado e então manipule-o conforme necessário:

```
Private Sub btn_EmpOK_Click()
Dim LastRow As Long, i As Integer
LastRow = Worksheets("Sheet2").Cells(Worksheets("Sheet2").Rows.Count, 1) _
.End(xlUp).Row + 1
Cells(LastRow, 1).Value = tb_EmpName.Value
'verifica o status de seleção dos itens em ListBox
For i = 0 To lb_EmpPosition.ListCount - 1
'se o item está selecionado, adiciona-o à planilha
    If lb_EmpPosition.Selected(i) = True Then
        Cells(LastRow, 2).Value = Cells(LastRow, 2).Value & _
        lb_EmpPosition.List(i) & ","
    End If
Next i
Cells(LastRow, 3).Value = tb_HireDate.Value
End Sub
```

A contagem dos itens em uma *caixa de listagem* começa em zero; assim, se utilizar a `ListCount`, você deverá subtrair um do resultado:

```
For i = 0 To lb_EmpPosition.ListCount - 1
```

Os eventos disponíveis para os controles `ListBox` e `ComboBox` são descritos na Tabela 10.3.

Tabela 10.3 Os eventos para os controles `ListBox` e `ComboBox`

Evento	Descrição
`AfterUpdate`	Ocorre depois que os dados do controle foram alterados pelo usuário.
`BeforeDragOver`	Ocorre enquanto o usuário arrasta e solta dados sobre o controle.
`BeforeDropOrPaste`	Ocorre antes de o usuário estar prestes a soltar ou colar dados no controle.
`BeforeUpdate`	Ocorre antes dos dados no controle ser alterados.
`Change`	Ocorre quando o valor do controle é alterado.
`Click`	Ocorre quando o usuário seleciona um valor na caixa de listagem ou de combinação.
`DblClick`	Ocorre quando o usuário dá um clique duplo no controle com o mouse.
`DropButtonClick`[1]	Ocorre quando a lista suspensa aparece depois que o usuário clica na seta suspensa da caixa de combinação ou pressiona F4 no teclado.
`Enter`	Ocorre antes de o controle receber o foco de outro controle no mesmo userform.
`Error`	Ocorre quando o controle encontra um erro e não pode retornar as informações sobre o erro.
`Exit`	Ocorre assim que o foco do controle passa para outro controle no mesmo userform.
`KeyDown`	Ocorre quando o usuário pressiona uma tecla no teclado.
`KeyPress`	Ocorre quando o usuário pressiona uma tecla ANSI. Uma tecla ANSI é um caractere digitável, como a letra *A*. Um caractere não-digitável seria, por exemplo, a tecla Tab.
`KeyUp`	Ocorre quando o usuário solta uma tecla no teclado.
`MouseDown`	Ocorre quando o usuário pressiona o botão do mouse dentro das bordas do controle.
`MouseMove`	Ocorre quando o usuário move o mouse dentro das bordas do controle.
`MouseUp`	Ocorre quando o usuário solta o botão do mouse dentro das bordas do controle.

[1] Controle *`ComboBox`* somente

Adicionando botões de opção a um userform

Os botões de opção são semelhantes a caixas de seleção porque podem ser utilizados para fazer uma seleção. Mas, diferentemente das caixas de seleção, eles podem ser facilmente configurados para permitir apenas uma seleção a partir de um grupo.

Utilizando a ferramenta Frame, desenhe um frame para separar o próximo conjunto de controles do outros controles no userform. O frame é utilizado para agrupar botões de opção, como mostrado na Figura 10.11.

Figura 10.11
Utilize um frame para agrupar botões de opção.

Os botões de opção têm uma propriedade `GroupName`. Se atribuir o mesmo nome de grupo, Prédios, a um conjunto de botões de opção, você faz com que eles funcionem coletivamente como um mecanismo alternador, para que apenas um botão no conjunto possa ser selecionado. Selecionar um botão de opção remove automaticamente a seleção dos outros botões no mesmo grupo ou frame. Para impedir esse comportamento, deixe a propriedade `GroupName` em branco ou insira outro nome.

DICA Para os usuários que preferem selecionar o rótulo do botão de opção em vez do próprio botão, adicione o código ao rótulo para desencadear o botão de opção.

```
Private Sub Lbl_Bldg1_Click()
Obtn_Bldg1.Value = True
End Sub
```

Os eventos disponíveis para os controles `OptionButton` e `Frame` são descritos na Tabela 10.4.

Tabela 10.4 Os eventos para os controles `OptionButton` e `Frame`

Evento	Descrição
`AfterUpdate`[1]	Ocorre depois que os dados do controle foram alterados pelo usuário.
`AddControl`[2]	Ocorre quando um controle é adicionado a um frame em um formulário em tempo de execução. Não executa em tempo de projeto ou na inicialização de userform.
`BeforeDragOver`	Ocorre enquanto o usuário arrasta e solta sobre o controle.
`BeforeDropOrPaste`	Ocorre antes de o usuário estar prestes a soltar ou colar dados no controle.
`BeforeUpdate`[1]	Ocorre antes de os dados no controle serem alterados.
`Change`[1]	Ocorre quando o valor do controle é alterado.
`Click`	Ocorre quando o usuário clica no controle com o mouse.
`DblClick`	Ocorre quando o usuário dá um clique duplo no controle com o mouse.
`Enter`	Ocorre antes de o controle receber o foco de outro controle no mesmo userform.
`Error`	Ocorre quando o controle encontra um erro e não pode retornar as informações sobre o erro.
`Exit`	Ocorre assim que o foco do controle passa para outro controle no mesmo userform.
`KeyDown`	Ocorre quando o usuário pressiona uma tecla no teclado.
`KeyPress`	Ocorre quando o usuário pressiona uma tecla ANSI. Uma tecla ANSI é um caractere digitável, como a letra *A*. Um caractere não-digitável seria, por exemplo, a tecla Tab.
`KeyUp`	Ocorre quando o usuário solta uma tecla no teclado.
`Layout`[2]	Ocorre quando o frame muda de tamanho.
`MouseDown`	Ocorre quando o usuário pressiona o botão do mouse dentro das bordas do controle.
`MouseMove`	Ocorre quando o usuário move o mouse dentro das bordas do controle.
`MouseUp`	Ocorre quando o usuário libera o botão do mouse dentro das bordas do controle.
`RemoveControl`[2]	Ocorre quando um controle é excluído de dentro do controle de frame.
`Scroll2`	Ocorre quando a caixa de barra de rolagem, se visível, é reposicionada.
`Zoom`[2]	Ocorre quando o valor de zoom é alterado.

[1] Controle *`OptionButton`* somente

[2] Controle *`Frame`* somente

Adicionando elementos gráficos a um userform

Uma listagem em um formulário pode ser ainda mais útil se um elemento gráfico correspondente for adicionado ao formulário. O código a seguir exibe a fotografia que corresponde ao funcionário selecionado na caixa de listagem:

```
Private Sub lb_EmpName_Change()
Dim EmpFound As Range
With Range("EmpList")
    Set EmpFound = .Find(lb_EmpName.Value)
    If EmpFound Is Nothing Then
        MsgBox ("Funcionário não localizado!")
        lb_EmpName.Value = ""
        Exit Sub
    Else
        With Range(EmpFound.Address)
            tb_EmpPosition = .Offset(0, 1)
            tb_HireDate = .Offset(0, 2)
            On Error Resume Next
            Img_Employee.Picture = LoadPicture _
        ("fig1012a.bmp")
            On Error GoTo 0
        End With
    End If
End With
```

Os eventos disponíveis para controles `Graphic` são descritos na Tabela 10.5.

Tabela 10.5 Eventos para controles `Graphic`

Evento	Descrição
`BeforeDragOver`	Ocorre enquanto o usuário arrasta e solta os dados sobre o controle.
`BeforeDropOrPaste`	Ocorre antes de o usuário estar prestes a soltar ou colar dados no controle.
`Click`	Ocorre quando o usuário clica na imagem com o mouse.
`DblClick`	Ocorre quando o usuário dá um clique duplo na imagem com o mouse.
`Error`	Ocorre quando o controle encontra um erro e não consegue retornar informações sobre o erro.
`MouseDown`	Ocorre quando o usuário pressiona o botão do mouse dentro das bordas da imagem.
`MouseMove`	Ocorre quando o usuário move o mouse dentro das bordas da imagem.
`MouseUp`	Ocorre quando o usuário solta o botão do mouse dentro das bordas do controle.

Utilizando um botão de rotação em um userform

Da maneira como ele está, o campo Data de Contratação permite que o usuário insira a data em qualquer formato: 1/1/1 ou janeiro 1, 2001. Essa possível inconsistência pode criar problemas mais tarde se você tiver de utilizar ou procurar datas. A solução? Force os usuários a inserir datas de maneira unificada.

Os botões de rotação permitem ao usuário incrementar/decrementar ao longo de uma série de números. Dessa maneira, o usuário é forçado a inserir números em vez de texto.

Desenhe um botão de rotação para uma entrada de Mês no formulário. Em Propriedades, configure Min como 1 (para janeiro) e Max como 12 (para dezembro). Na propriedade Valor, insira 1, o primeiro mês. Em seguida, desenhe uma caixa de texto ao lado do botão de rotação. Essa é a caixa de texto que refletirá o valor do botão de rotação. (Rótulos também podem ser utilizados.)

```
Private Sub SpBtn_Month_Change()
tb_Month.Value = SpBtn_Month.Value
End Sub
```

Termine de construir o formulário. Utilize um Min de 1 e um Max de 31 para dias; Min de 1900 e um Max de 2100 para anos:

```
Private Sub btn_EmpOK_Click()
Dim LastRow As Long, i As Integer
LastRow = Worksheets("Sheet2").Cells(Worksheets("Sheet2").Rows.Count, 1) _
```

```
.End(xlUp).Row + 1
Cells(LastRow, 1).Value = tb_EmpName.Value
For i = 0 To lb_EmpPosition.ListCount - 1
    If lb_EmpPosition.Selected(i) = True Then
        Cells(LastRow, 2).Value = Cells(LastRow, 2).Value & _
        lb_EmpPosition.List(i) & ","
    End If
Next i
'Concatena os valores das caixas de texto para criar a data
Cells(LastRow, 3).Value = tb_Month.Value & "/" & tb_Day.Value & _
    "/" & tb_Year.Value
End Sub
```

Os eventos disponíveis para controles `SpinButton` são descritos na Tabela 10.6.

Tabela 10.6 Eventos para controles `SpinButton`

Evento	Descrição
`AfterUpdate`	Ocorre depois que os dados do controle foram alterados pelo usuário.
`BeforeDragOver`	Ocorre enquanto o usuário arrasta e solta dados sobre o controle.
`BeforeDropOrPaste`	Ocorre antes de o usuário estar prestes a soltar ou colar dados no controle.
`BeforeUpdate`	Ocorre antes dos dados no controle ser alterados.
`Change`	Ocorre quando o valor do controle é alterado.
`DblClick`	Ocorre quando o usuário dá um clique duplo no controle.
`Enter`	Ocorre antes de o controle receber o foco de outro controle no mesmo userform.
`Error`	Ocorre quando o controle encontra um erro e não pode retornar as informações sobre o erro.
`Exit`	Ocorre assim que o foco do controle passa para outro controle no mesmo userform.
`KeyDown`	Ocorre quando o usuário pressiona uma tecla no teclado.
`KeyPress`	Ocorre quando o usuário pressiona uma tecla ANSI. Uma tecla ANSI é um caractere digitável, como a letra *A*. Um caractere não-digitável seria, por exemplo, a tecla Tab.
`KeyUp`	Ocorre quando o usuário solta uma tecla no teclado.
`SpinDown`	Ocorre quando o usuário clica no botão de rotação inferior ou esquerdo, diminuindo o valor.
`SpinUp`	Ocorre quando o usuário clica no botão seletor superior ou direito, aumentando o valor.

Utilizando o controle `MultiPage` para combinar formulários

O controle `MultiPage` fornece uma maneira elegante de organizar vários formulários. Em vez de um formulário para informações pessoais sobre um funcionário e outro para informações relacionadas ao trabalho, combine as informações em um formulário de várias páginas, como mostrado na Figuras 10.12 e 10.13.

Figura 10.12 Utilize o controle `MultiPage` para combinar múltiplos formulários. Essa é a primeira página do formulário.

Figura 10.13
A segunda página.

> **DICA** Formulários de várias páginas devem ser planejados desde o início — adicioná-los depois que o restante do formulário é criado não é uma tarefa fácil. Se mais tarde você decidir que precisa de um formulário de múltiplas páginas, insira um novo formulário, desenhe as múltiplas páginas e copie-e-cole os controles a partir dos outros formulários no novo formulário.

> **NOTA** Diferentemente dos outros controles, você não pode clicar com o botão direito do mouse no controle `MultiPage` e visualizar o código. Em vez disso, selecione o controle e pressione F7 no teclado ou acesse Exibir, Código.

Você pode modificar uma página clicando nela com o botão direito e abrindo o menu de opções: Inserir Uma Nova Página, Excluir a Página em que Você Clicou Com o Botão Direito do Mouse, Renomear Uma Página ou Mover Uma página.

Diferentemente dos vários outros controles, nos quais a propriedade `Value` armazena um valor inserido ou selecionado pelo usuário, a propriedade `Value` do controle `MultiPage` armazena o número da página ativa, iniciando em zero. Por exemplo, se você tiver um formulário de cinco páginas e quiser ativar a quarta página, faça isto:

```
MultiPage1.Value = 4
```

Se houver um controle que você quer que seja compartilhado por todas as páginas — como o botão Salvar ou Cancelar — posicione o controle no userform principal em vez de nas páginas individuais, como mostrado na Figura 10.14.

Figura 10.14
Posicione os controles comuns no userform principal.

Os eventos disponíveis para controles `MultiPage` são descritos na Tabela 10.7.

Tabela 10.7 Eventos para o controle `MultiPage`

Evento	Descrição
`AddControl`	Ocorre quando um controle é adicionado a uma página do controle `MultiPage`. Não executa em tempo de projeto ou na inicialização de userform.
`BeforeDragOver`	Ocorre enquanto o usuário arrasta e solta dados em uma página do controle `MultiPage`.
`BeforeDropOrPaste`	Ocorre antes de o usuário estar prestes a soltar ou colar dados em uma página do controle `MultiPage`.
`Change`	Ocorre quando o usuário muda as páginas de um controle de multipágina.
`Click`	Ocorre quando o usuário clica em uma página do controle `MultiPage`.
`DblClick`	Ocorre quando o usuário dá um clique duplo em uma página do controle `MultiPage` com o mouse.

Tabela 10.7 Eventos para o controle `MultiPage` (cont.)

Evento	Descrição
`Enter`	Ocorre antes de a multipágina receber o foco de outro controle no mesmo userform.
`Error`	Ocorre quando o controle `MultiPage` encontra um erro e não pode retornar informações sobre o erro.
`Exit`	Ocorre assim que a multipágina perde foco para outro controle no mesmo userform.
`KeyDown`	Ocorre quando o usuário pressiona uma tecla no teclado.
`KeyPress`	Ocorre quando o usuário pressiona uma tecla ANSI. Uma tecla ANSI é um caractere digitável, como a letra *A*. Um caractere não-digitável seria, por exemplo, a tecla Tab.
`KeyUp`	Ocorre quando o usuário solta uma tecla no teclado.
`MouseDown`	Ocorre quando o usuário pressiona o botão do mouse dentro das bordas do controle.
`MouseMove`	Ocorre quando o usuário move o mouse dentro das bordas do controle.
`MouseUp`	Ocorre quando o usuário libera o botão do mouse dentro das bordas do controle.
`RemoveControl`	Ocorre quando um controle é removido de uma página da multipágina.
`Scroll`	Ocorre quando a caixa de barra de rolagem, se visível, é reposicionada.
`Zoom`	Ocorre quando o valor de zoom é alterado.

Verificando a entrada de campo

Mesmo se os usuários forem informados de que devem preencher todos os campos do formulário, não há como forçá-los a fazer isso — exceto com um formulário eletrônico. Como programador, você pode assegurar que todos os campos necessários sejam preenchidos, não permitindo que o usuário continue até que todos os requisitos sejam atendidos:

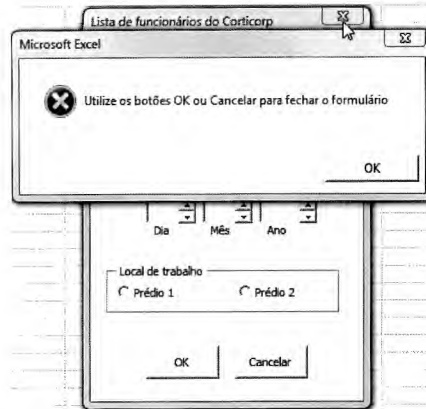
```
If tb_EmpName.Value = "" Then
    frm_AddEmp.Hide
    MsgBox ("Insira um nome de funcionário")
    frm_AddEmp.Show
    Exit Sub
End If
```

Fechamento inválido de janela

Os userforms criados no Editor do VB não são diferentes das janelas comuns: eles também incluem o botão fechar (X) no canto superior direito. Embora não seja errado utilizar esse botão, ele pode causar problemas, dependendo do objetivo do userform. Nesses casos, talvez você queira controlar o que acontecerá se o usuário pressionar o botão. Utilize o evento `QueryClose` do userform para descobrir que método é utilizado para fechar o formulário e para codificar uma ação apropriada:

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
If CloseMode = vbFormControlMenu Then
    MsgBox "Utilize os botões OK ou Cancelar para fechar o formulário", vbCritical
    Cancel = True
End If
End Sub
```

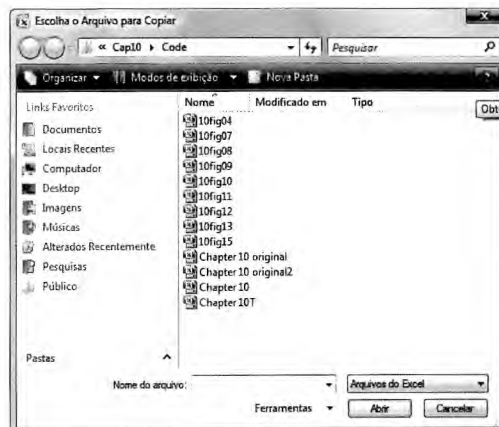
Depois que tiver conhecimento do método que o usuário utilizou para tentar fechar o formulário, você poderá criar uma caixa de mensagem semelhante à da Figura 10.15, alertando-o de que o método é inválido.

Figura 10.15
Controle o que acontece quando o usuário clica no botão X.

O evento `QueryClose` pode ser desencadeado de três outras maneiras:

- `vbFormCode` — A instrução `Unload` foi utilizada.
- `vbAppWindows` — O Windows é desligado.
- `vbAppTaskManager` — O aplicativo foi desligado pelo Task Manager.

Obtendo um nome de arquivo

Uma das interações mais comuns com clientes é quando você precisa que eles especifiquem um caminho e nome de arquivo. O Excel VBA tem uma função predefinida para exibir a caixa de diálogo Abrir Arquivo, como mostrado na Figura 10.16. O cliente navega e seleciona um arquivo. Quando ele escolhe o botão Abrir, o Excel VBA não abre o arquivo, mas retorna o arquivo selecionado para você.

Figura 10.16
Utilize a caixa de diálogo Abrir Arquivo para permitir que o usuário selecione um arquivo.

```
Sub SelectFile()
'Pergunta que arquivo copiar
x = Application.GetOpenFilename( _
    FileFilter:="Arquivos do Excel (*.xls*), *.xls*", _
    Title:="Escolha o arquivo para copiar", MultiSelect:=False)

'Verifica se nenhum arquivo foi selecionado
If x = "False" Then Exit Sub

MsgBox "Você selecionou " & x
End Sub
```

O código acima permitirá que o cliente selecione um arquivo. Se você quiser que ele especifique múltiplos arquivos, utilize este código:

```
Sub ManyFiles()
Dim x As Variant

x = Application.GetOpenFilename( _
    FileFilter:="Arquivos do Excel (*.xls*), *.xls*", _
```

```
    Title:="Escolha os arquivos", MultiSelect:=True)

'Verifica se nenhum arquivo foi selecionado
On Error Resume Next
If x = "False" Then Exit Sub
On Error GoTo 0

For i = 1 To UBound(x)
    MsgBox "Você selecionou " & x(i)
Next i
End Sub
```

De modo semelhante, você pode utilizar Application.GetSaveAsFileName para localizar o caminho e o nome de arquivo que devem ser utilizados para salvar um arquivo.

Próximos passos

Agora que você viu como trabalhar com userforms, o próximo capítulo examina gráficos. Você aprenderá como o gráfico de planilha se tornou um recurso altamente personalizável, capaz de tratar grandes quantidades de dados.

Criando gráficos

11

Gráficos no Excel 2007

A Microsoft deu o ambicioso salto para tentar substituir todo o mecanismo gráfico com o qual contou nos últimos 15 anos. Embora eu aprecie o desejo da Microsoft de fazer algo completamente novo, não houve tempo suficiente de desenvolvimento para concluir a tarefa. Isso nos deixa com três problemas:

- O código de todos os novos recursos não é compatível com as versões anteriores do Excel. Há um subconjunto irritante de recursos que funciona na versão atual do Excel, mas não funciona nas anteriores. Mas todos os bons recursos em gráfico são novos no Excel 2007; portanto, dificilmente algum código neste capítulo é compatível com o Excel 97–2003. Se você precisar escrever de código para criar gráficos do Excel 2003, pode usar os exemplos do Capítulo 10 do nosso livro *VBA and Macros for Microsoft Excel* (ISBN 978-0-7897-3129-0, Que Publishing). O arquivo de projeto daquele capítulo está disponível em www.mrexcel.com/vba2007data.html.
- A Microsoft conseguiu concluir algum trabalho no programa de gravação de macros para os novos recursos gráficos. O programa de gravação de macros pode gravar a maioria de ações nas guias Design e Layout, mas ignora completamente as ações na guia Formatar ou nas caixas de diálogo Formato. O código VBA está disponível para elementos gráficos de microformato, mas é preciso escrever o código para realizar ações na guia Formatar a partir do zero. Este livro serve como referência para ajudá-lo nessa empreitada. Você também deve ler a seção "Utilizando a janela Inspeção para descobrir configurações de objeto", mais adiante neste capítulo, para conhecer uma técnica para aprender o código VBA que não é gravado pelo programa de gravação de macros.
- Há bugs no mecanismo de gráficos do Excel 2007. Em alguns casos, editar a função `SÉRIE` faz o Excel travar. Em outros casos, a função `SÉRIE` nem aparecerá. Tente criar um gráfico abaixo da linha 1100. Percorrer o gráfico para baixo e para cima novamente o exibirá de maneiras estranhas. Para aliviar esse problema, procure um Office 2007 Service Release em 2008 e faça o download assim que estiver disponível.

NOVO Escrevendo código para os novos recursos gráficos do Excel 2007

Os gráficos foram completamente reescritos no Excel 2007. A maior parte do código do Excel 2003 continuará a funcionar no Excel 2007. Entretanto, se você gravar um código para tirar proveito dos novos recursos gráficos, esse código não será compatível com o Excel 2003.

A seguir, alguns novos métodos e recursos disponíveis no Excel 2007:

- **`ApplyLayout`** — Esse método aplica um dos layouts de gráfico disponíveis na guia Design.

NESTE CAPÍTULO

- `SetElement` — Esse método escolhe qualquer uma das opções de elemento predefinido da guia Layout.
- `ChartFormat` — Esse objeto permite alterar o preenchimento, o brilho, a linha, o reflexo, a sombra, a suavidade da borda ou o formato 3-D da maioria dos elementos gráficos individuais. Esse método é semelhante às configurações na guia Formatar.
- `AddChart` — Esse método permite inserir um gráfico em uma planilha existente.

Referenciando gráficos e objetos gráficos no código VBA

Se você voltar o suficiente na história do Excel, descobrirá que todos os gráficos eram criados como suas próprias planilhas de gráfico. Então, em meados de 1990, o Excel ganhou a surpreendente capacidade de incorporar um gráfico diretamente em uma planilha existente. Isso permitiu criar um relatório com tabelas de números e gráficos em uma mesma página, algo que nem nos damos conta hoje em dia.

Essas duas maneiras diferentes de lidar com gráficos exigiram que se aprendesse a lidar com dois modelos de objeto separados para gráficos. Quando um gráfico estiver em sua própria planilha de gráfico independente, ele é um objeto Chart. Quando um gráfico estiver incorporado em uma planilha, você está lidando com um objeto ChartObject. O Excel 2007 introduz um terceiro ramo evolutivo, porque os objetos em uma planilha também são membros da coleção Formas.

No Excel 2003, para referenciar a cor da área de gráfico de um gráfico incorporado, você teria de referir-se ao gráfico desta maneira:

```
Worksheets("Jan").ChartObjects("Chart 1").Chart.ChartArea.Interior.ColorIndex = 4
```

Em vez disso, no Excel 2007, você pode utilizar a coleção Formas:

```
Worksheets("Jan").Shapes("Chart 1").Chart.ChartArea.Interior.ColorIndex = 4
```

Em qualquer versão do Excel, se um gráfico estiver em sua própria planilha de gráfico, você não têm de especificar o contêiner; pode simplesmente referenciar o objeto Chart:

```
Sheets("Chart1").ChartArea.Interior.ColorIndex = 4
```

Criando um gráfico

Nas primeiras versões do Excel, utilizava-se o comando `Charts.Add` para adicionar um novo gráfico. Você especificava os dados originais, o tipo de gráfico e se o gráfico devia estar em uma nova planilha ou incorporado em uma planilha existente. As três primeiras linhas do código a seguir criam um gráfico de colunas agrupadas em uma nova planilha de gráfico. A quarta linha move novamente o gráfico para ser um objeto incorporado em `Sheet1`:

```
Charts.Add
ActiveChart.SetSourceData Source:=Worksheets("Sheet1").Range("A1:E4")
ActiveChart.ChartType = xlColumnClustered
ActiveChart.Location Where:=xlLocationAsObject, Name:="Sheet1"
```

Se planeja compartilhar suas macros com as pessoas que ainda utilizam o Excel 2003, você deve utilizar o método `Charts.Add`. Mas se seu aplicativo for executar apenas no Excel 2007, pode utilizar o novo método AddChart. O código do método `AddChart` pode ser tão simples quanto o seguinte:

```
'Crie gráfico na planilha atual
ActiveSheet.Shapes.AddChart.Select
ActiveChart.SetSourceData Source:=Range("A1:E4")
ActiveChart.ChartType = xlColumnClustered
```

Ou, você pode especificar o tipo de gráfico, o tamanho e a localização como parte do método `AddChart`, como descrito na próxima seção.

Especificando o tamanho e o local de um gráfico

O método `AddChart` tem parâmetros adicionais que podem ser usados para especificar o tipo e o tamanho do gráfico e o local dele na planilha.

O local e o tamanho de um gráfico são especificados em pontos (72 pontos = 1 polegada). Por exemplo, o parâmetro `Top` requer o número de pontos do topo da linha 1 à borda superior da planilha.

O código a seguir cria um gráfico que abrange aproximadamente o intervalo C11:J30:

```
Sub SpecifyLocation()
    Dim WS As Worksheet
```

```
    Set WS = Worksheets("Sheet1")
    WS.Shapes.AddChart(xlColumnClustered, _
        Left:=100, Top:=150, _
        Width:=400, Height:=300).Select
    ActiveChart.SetSourceData Source:=WS.Range("A1:E4")
End Sub
```

Isso exigiria bastante tentativa e erro para descobrir aleatoriamente a distância exata em pontos para fazer com que um gráfico se alinha com determinada célula. Felizmente, você pode perguntar ao VBA qual é a distância em pontos para certa célula. Se solicitar a propriedade `Left` de qualquer célula, você localiza a distância para o canto superior esquerdo dessa célula. Também é possível solicitar a largura ou a altura de um intervalo. Por exemplo, o código a seguir cria um gráfico exatamente em C11:J30:

```
Sub SpecifyExactLocation()
    Dim WS As Worksheet
    Set WS = Worksheets("Sheet1")
    WS.Shapes.AddChart(xlColumnClustered, _
        Left:=WS.Range("C11").Left, _
        Top:=WS.Range("C11").Top, _
        Width:=WS.Range("C11:J11").Width, _
        Height:=WS.Range("C11:C30").Height).Select
    ActiveChart.SetSourceData Source:=WS.Range("A1:E4")
End Sub
```

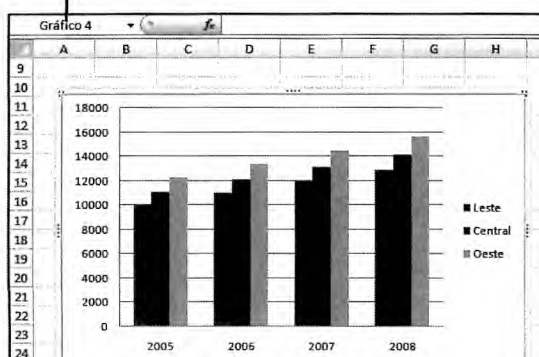
Nesse caso, você não está movendo o local do objeto Chart; em vez disso, está movendo o local do contêiner do gráfico. No Excel 2007, esse é o objeto ChartObject ou Shape. Se tentar mudar a localização real do gráfico, você o move dentro do contêiner. Como você pode de fato mover alguns pontos da área do gráfico para qualquer direção dentro do contêiner, o código executará, mas você não terá os resultados desejados.

Para mover um gráfico que já foi criado, você pode referenciar ChartObject ou Shape e alterar as propriedades `Top`, `Left`, `Width` e `Height` conforme mostrado na seguinte macro:

```
Sub MoveAfterTheFact()
    Dim WS As Worksheet
    Set WS = Worksheets("Sheet1")
    With WS.ChartObjects("Chart 9")
        .Left = WS.Range("C21").Left
        .Top = WS.Range("C21").Top
        .Width = WS.Range("C1:H1").Width
        .Height = WS.Range("C21:C25").Height
    End With
End Sub
```

Referência posterior a um gráfico específico

Ao criar um novo gráfico, ele recebe um nome seqüencial, como Gráfico 1. Se selecionar um gráfico e depois observar a caixa Nome, você verá o nome do gráfico. Na Figura 11.1, o nome do gráfico é Gráfico 4. Isso não significa que há quatro gráficos na planilha. Nesse caso particular, gráficos individuais foram criados e excluídos.

Figura 11.1
Você pode selecionar um gráfico e examinar a Caixa de nome para localizar o nome dele.

Isso significa que, dependendo do dia em que sua macro executa, o objeto Chart pode ter um nome diferente. Se precisar referenciar o gráfico mais tarde na macro, talvez você tenha de selecionar outras células e o gráfico não estará mais ativo, e pode perguntar ao VBA o nome do gráfico e armazená-lo em uma variável para uso posterior, como mostrado aqui:

```
Sub RememberTheName()
    Dim WS As Worksheet
    Set WS = Worksheets("Sheet1")
    WS.Shapes.AddChart(xlColumnClustered, _
        Left:=WS.Range("C11").Left, _
        Top:=WS.Range("C11").Top, _
        Width:=WS.Range("C11:J11").Width, _
        Height:=WS.Range("C11:C30").Height _
        ).Select
    ActiveChart.SetSourceData Source:=WS.Range("A1:E4")
    ' Lembra do nome de uma variável
    ThisChartObjectName = ActiveChart.Parent.Name
    ' Mais linhas de código...
    ' Mais tarde, então, você precisa reatribuir o gráfico na macro
    With WS.Shapes(ThisChartObjectName)
        .Chart.SetSourceData Source:=WS.Range("A20:E24"), PlotBy:=xlColumns
        .Top = WS.Range("C26").Top
    End With
End Sub
```

Na macro anterior, a variável `ThisChartObjectName` contém o nome do objeto Chart. Esse método funciona muito bem se suas alterações acontecerem mais tarde na mesma macro. Mas depois que a macro termina de executar, a variável estará fora de escopo e você não conseguirá acessar o nome mais tarde.

Se quiser se lembrar do nome de um gráfico, você pode armazená-lo em uma célula mais isolada da planilha. Aqui, a primeira macro armazena o nome na célula Z1 e a segunda macro modifica mais tarde o gráfico usando o nome armazenado na célula Z1:

```
Sub StoreTheName()
    Dim WS As Worksheet
    Set WS = Worksheets("Sheet1")
    WS.Shapes.AddChart(xlColumnClustered, _
        Left:=WS.Range("C11").Left, _
        Top:=WS.Range("C11").Top, _
        Width:=WS.Range("C11:J11").Width, _
        Height:=WS.Range("C11:C30").Height _
        ).Select
    ActiveChart.SetSourceData Source:=WS.Range("A1:E4")
    Range("Z1").Value = ActiveChart.Parent.Name
End Sub
```

Depois de a macro anterior armazenar o nome na célula Z1, a macro seguinte utiliza o valor em Z1 para descobrir qual macro alterar:

```
Sub ChangeTheChartLater()
    Dim WS As Worksheet
    Set WS = Worksheets("Sheet1")
    MyName = WS.Range("Z1").Value
    With WS.Shapes(MyName)
        .Chart.SetSourceData Source:=WS.Range("A20:E24"), PlotBy:=xlColumns
        .Top = WS.Range("C26").Top
    End With
End Sub
```

Se precisar modificar um gráfico preexistente — como um que você não criou — e houver apenas um gráfico na planilha, você pode usar esta linha de código:

```
WS.ChartObjects(1).Chart.Interior.ColorIndex = 4
```

Se houver muitos gráficos e você precisar localizar aquele com o canto superior esquerdo localizado na célula A4, você pode percorrer todos os objetos Chart até encontrar aquele do local certo, como este:

```
For each Cht in ActiveSheet.ChartObjects
    If Cht.TopLeftCell.Address = "$A$4" then
        Cht.Interior.ColorIndex = 4
    end if
Next Cht
```

NOVO Gravando comandos das guias Layout ou Design

Com os gráficos no Excel 2007, há três níveis de alterações. As configurações de gráfico globais — o tipo e o estilo do gráfico — residem na guia Design. As seleções das configurações de elementos predefinidos aparecem na guia Layout. Você faz microalterações usando a guia Formatar.

O programa de gravação de macros no Excel 2007 faz um ótimo trabalho de gravar alterações nas guias Design e Layout; então, se você precisar fazer certas alterações, pode gravar rapidamente uma macro e copiar o código.

Especificando um tipo de gráfico predefinido

Há 73 tipos de gráficos predefinidos no Excel 2007. Para mudar um gráfico para um dos 73 tipos, você utiliza a propriedade `ChartType`. Essa propriedade pode ser aplicada a um gráfico ou uma série dentro de um gráfico. Eis um exemplo que muda o tipo do gráfico inteiro:

```
ActiveChart.ChartType = xlBubble
```

Para definir a segunda série de um gráfico como um gráfico de linha, você utiliza isto:

```
ActiveChart.Series(2).ChartType = xlLine
```

A Tabela 11.1 relaciona as 73 constantes de tipo de gráfico que podem ser utilizadas para criar vários gráficos. A seqüência da Tabela 11.1 corresponde à dos gráficos na caixa de diálogo Tipo de Gráfico.

Tabela 11.1 Tipos de gráfico para uso no VBA

Tipo de gráfico	Constante
Colunas agrupadas	xlColumnClustered
Colunas empilhadas	xlColumnStacked
Colunas 100% empilhadas	xlColumnStacked100
Colunas agrupadas em 3-D	xl3DColumnClustered
Colunas empilhadas em 3-D	xl3DColumnStacked
Colunas 100% empilhadas em 3-D	xl3DColumnStacked100
Colunas 3D	xl3DColumn
Barras agrupadas	xlCylinderColClustered
Barras empilhadas	xlCylinderColStacked
Barras 100% empilhadas	xlCylinderColStacked100
Barras em 3-D	xlCylinderCol
Cones agrupados	xlConeColClustered
Cones empilhados	xlConeColStacked
Cones 100% empilhados	xlConeColStacked100
Cone em 3-D	xlConeCol
Pirâmide agrupada	xlPyramidColClustered
Pirâmide empilhada	xlPyramidColStacked

Tabela 11.1 Tipos de gráfico para uso no VBA (*cont.*)

Tipo de gráfico	Constante
Pirâmide 100% empilhada	xlPyramidColStacked100
Pirâmide em 3-D	xlPyramidCol
Linha	xlLine
Linha empilhada	xlLineStacked
Linha 100% empilhada	xlLineStacked100
Linha com marcadores	xlLineMarkers
Linha empilhada com marcadores	xlLineMarkersStacked
Linha 100% empilhada com marcadores	xlLineMarkersStacked100
Linhas 3D	xl3DLine
Pizza	xlPie
Pizza em 3-D	xl3DPie
Pizza de Pizza	xlPieOfPie
Pizza destacada	xlPieExploded
Pizza destacada em 3-D	xl3DPieExploded
Barra de pizza	xlBarOfPie
Barra agrupada	xlBarClustered
Barra empilhada	xlBarStacked
Barra 100% empilhada	xlBarStacked100
Barra agrupada em 3-D	xl3DBarClustered
Barra empilhada em 3-D	xl3DBarStacked
Barra 100% empilhada em 3-D	xl3DBarStacked100
Cilindro horizontal agrupado	xlCylinderBarClustered
Cilindro horizontal empilhado	xlCylinderBarStacked
100% Cilindro Horizontal Empilhado	xlCylinderBarStacked100
Cone horizontal agrupado	xlConeBarClustered
Cone horizontal empilhado	xlConeBarStacked
Cone Horizontal 100% empilhado	xlConeBarStacked100
Pirâmide horizontal agrupada	xlPyramidBarClustered

Tipo de gráfico	Constante
Pirâmide horizontal empilhada	xlPyramidBarStacked
Pirâmide Horizontal 100% empilhada	xlPyramidBarStacked100
Área	xlArea
Área empilhada	xlAreaStacked
Área 100% empilhada	xlAreaStacked100
Área 3-D	xl3DArea
Área empilhada em 3-D	xl3DAreaStacked
Área 100% empilhada em 3-D	xl3DAreaStacked100
Dispersão com marcadores apenas	xlXYScatter
Dispersão com linhas suaves e marcadores	xlXYScatterSmooth
Dispersão com linhas suaves	xlXYScatterSmoothNoMarkers
Dispersão com linhas retas e marcadores	xlXYScatterLines
Dispersão com linhas retas	xlXYScatterLinesNoMarkers
Gráfico de ações alta-baixa-fechamento	xlStockHLC
Gráfico de ações abertura–alta–baixa–fechamento	xlStockOHLC
Gráfico de ações volume–alta–baixa–fechamento	xlStockVHLC
Gráfico de ações volume–abertura–alta–baixa–fechamento	xlStockVOHLC
Superfície 3-D	xlSurface
Superfície 3-D delineada	xlSurfaceWireframe
Contorno	xlSurfaceTopView
Contorno delineado	xlSurfaceTopViewWireframe
Rosca	xlDoughnut
Rosca destacada	xlDoughnutExploded
Bolhas	xlBubble
Bolhas com um efeito 3-D	xlBubble3DEffect
Radar	xlRadar
Radar com marcadores	xlRadarMarkers
Radar preenchido	xlRadarFilled

Especificando um tipo de gráfico de modelo

Essa é uma ótima técnica para economizar tempo quando você está criando um gráfico com grande quantidade de formatação personalizada.

Uma *macro* VBA pode aproveitar um modelo de gráfico personalizado, contanto que você planeje distribuir o modelo de gráfico personalizado para cada pessoa que executará sua macro.

No Excel 2007, você salva tipos de gráficos personalizados como arquivos `.crtx` e os armazena na pasta `%appdata%\Microsoft\Templates\Charts\`.

Para aplicar um tipo de gráfico personalizado, utilize o seguinte:

```
ActiveChart.ApplyChartTemplate ("MyChart.crtx")
```

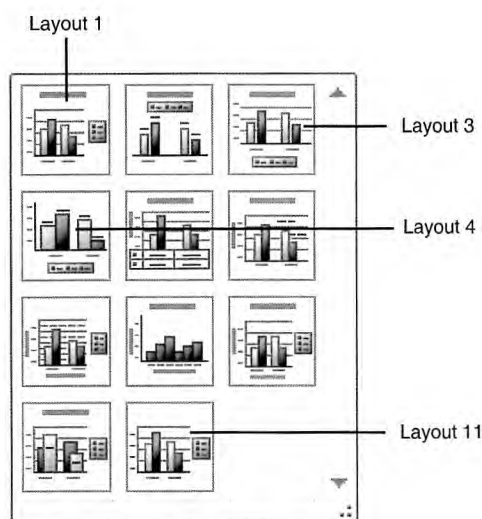
Se o modelo de gráfico não existir, o VBA retornará um erro. Se quiser que o Excel simplesmente continue sem exibir um erro de depuração, você pode desativar uma rotina de tratamento de erro antes do código e reativá-la quando terminar. Eis como fazer isso:

```
On Error Resume Next
ActiveChart.ApplyChartTemplate ("MyChart.crtx")
On Error GoTo 0 'O caractere final é um zero
```

Alterando o estilo e o layout de um gráfico

Dois grupos — o grupo Layout de Gráfico e o grupo Estilos de Gráfico — compõem a parte principal da guia Design.

O grupo Layout de Gráfico oferece de 4 a 12 combinações de elementos gráficos. Essas combinações são diferentes para os vários tipos de gráfico. Quando você acessar o grupo exibido na Figura 11.2, verá as Dicas de Tela para os layouts que mostram os layouts nomeados criativamente de Layout 1 a Layout 11.

Figura 11.2
Os layouts predefinidos são numerados de 1 a 11. Para outros tipos de gráficos, é possível ter de 4 a 12 layouts.

Para aplicar um dos layouts predefinidos em uma macro, é preciso utilizar o método `ApplyLayout` com um número de 1 a 12 para corresponder aos layouts predefinidos. O código a seguir aplicará Layout 1 ao gráfico ativo:

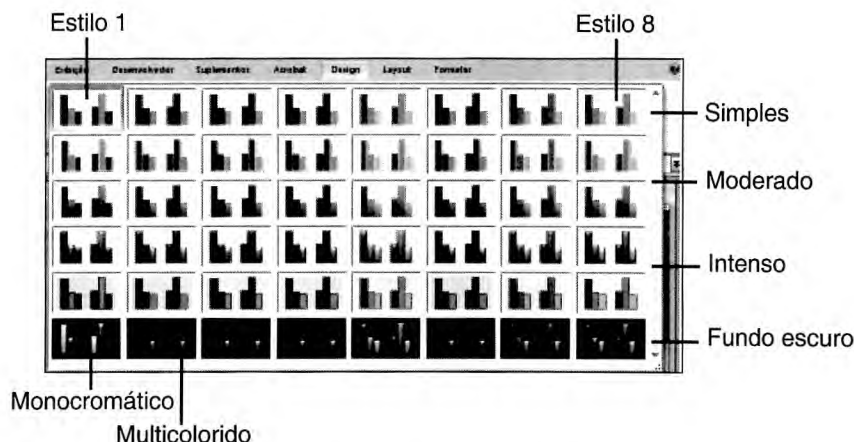
```
ActiveChart.ApplyLayout 1
```

ATENÇÃO

Embora os gráficos de linha ofereçam 12 layouts predefinidos, outros tipos de gráficos, como o radar, oferecem apenas quatro layouts predefinidos. Se você tentar aplicar um número de layout maior do que os disponíveis para o tipo de gráfico que estiver fazendo, o Excel retornará um erro de tempo de execução 5. A menos que você tenha acabado de criar o gráfico ativo na mesma macro, existe sempre a possibilidade de a pessoa que executa a macro mudar os gráficos de linha para gráficos de radar; por essa razão, inclua algum tratamento de erro antes de utilizar o comando `ApplyLayout`.

Evidentemente, para utilizar um layout predefinido com eficiência, você deve ter realmente feito um gráfico à mão e localizado um layout de que gosta.

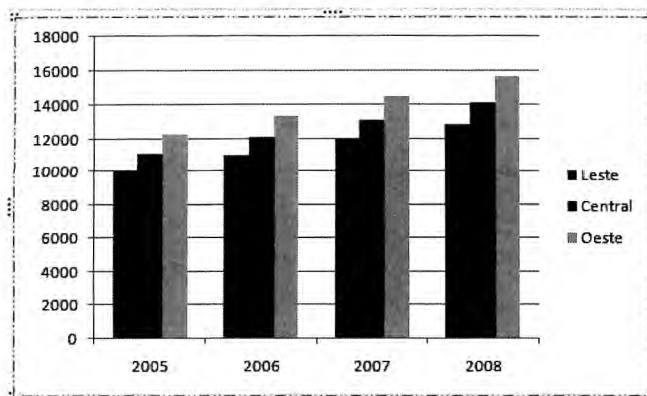
Como mostrado na Figura 11.3, o grupo *Estilos* de Gráfico contém 48 estilos. Esses estilos também são *numerados* seqüencialmente, com os estilos de 1 a 8 na linha 1, os estilos de 9 a 16 na linha 2, e assim por diante. Na verdade, esses estilos seguem um pequeno padrão:

Figura 11.3
Os estilos predefinidos são numerados de 1 a 48.

- Os estilos 1, 9, 17, 25, 33 e 41 (isto é, os estilos na coluna 1) são monocromáticos.
- Os estilos 2, 10, 18, 26, 34 e 42 (isto é, os estilos na coluna 2) utilizam cores diferentes para cada ponto.
- Todos os outros estilos usam tons de uma determinada cor de tema.
- Os estilos de 1 a 8 são simples.
- Os estilos de 17 a 24 utilizam efeitos moderados.
- Os estilos de 33 a 40 têm efeitos intensos.
- Os estilos de 41 a 48 aparecem em um fundo escuro.

Se você for misturar estilos em uma única pasta de trabalho, permaneça dentro de uma única linha ou coluna do grupo.
Para aplicar um estilo a um gráfico, utilize a propriedade `ChartStyle`, atribuindo-lhe um valor de `1` a `48`:

```
ActiveChart.ChartStyle = 1
```

A propriedade `ChartStyle` muda as cores do gráfico. Entretanto, várias alterações de formatação feitas a partir da guia Formatar não são sobrescritas quando se altera a propriedade `ChartStyle`. Por exemplo, na Figura 11.4, à segunda série havia sido aplicado um brilho e à terceira série, um vidro claro chanfrado. A execução do código anterior não eliminou essa formatação.

Figura 11.4
Configurar a propriedade `ChartStyle` não anula todas as configurações.

Para tirar qualquer formatação anterior, utilize o método `ClearToMatchStyle`:

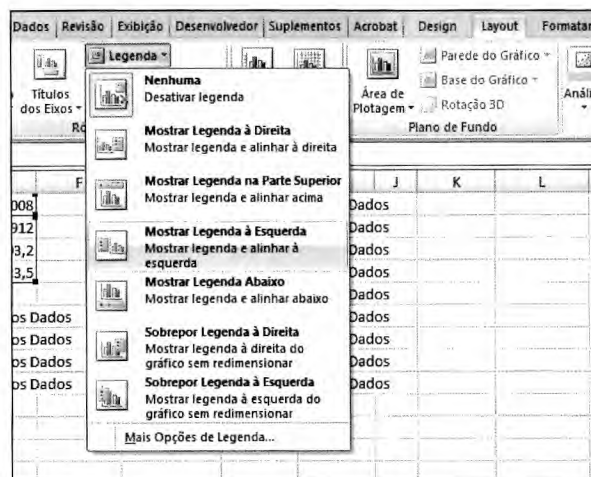
```
ActiveChart.ChartStyle = 1
ActiveChart.ClearToMatchStyle
```

Utilizando `SetElement` para imitar alterações na guia Layout

A guia Layout contém várias configurações predefinidas. A Figura 11.5 mostra alguns dos itens do menu predefinido para a guia Legenda. *Há* menus semelhantes para cada um dos ícones na figura.

Se você utilizar um item de menu predefinido para alterar os títulos, a legenda, os rótulos, os eixos, as linhas de grade ou o plano de fundo, provavelmente isso será tratado no código que utiliza o método `SetElement`, que é novo no Excel 2007.

`SetElement` não funciona com Mais Opções na parte inferior de cada menu. Esse método também não funciona com o botão Rotação 3-D. Exceto isso, você pode utilizar `SetElement` para alterar tudo nos grupos Rótulos, Eixos, Segundo Plano e Análise.

Figura 11.5
Há menus predefinidos semelhantes a esse para cada ícone. Se sua escolha estiver no menu, o código VBA utiliza o método `SetElement`.

O programa de gravação de macros sempre funciona para as configurações predefinidas na guia Layout. Se não quiser procurar a constante adequada neste livro, você sempre pode gravar uma macro rapidamente.

O método `SetElement` é seguido por uma constante que especifica o item do menu que será selecionado. Por exemplo, se quiser escolher Mostra Legenda à Esquerda, você pode utilizar esse código:

```
ActiveChart.SetElement msoElementLegendLeft
```

A Tabela 11.2 mostra todas as constantes disponíveis que podem ser utilizadas com o método SetElement. Elas estão quase na mesma ordem em que aparecem na guia Layout.

Tabela 11.2 Constantes disponíveis com SetElement

Ícone da guia layout	Constante do elemento gráfico
Título do gráfico	`msoElementChartTitleNone`
Título do gráfico	`msoElementChartTitleCenteredOverlay`
Título do gráfico	`msoElementChartTitleAboveChart`
Títulos dos eixos	`msoElementPrimaryCategoryAxisTitleNone`
Títulos dos eixos	`msoElementPrimaryCategoryAxisTitleBelowAxis`
Títulos dos eixos	`msoElementPrimaryCategoryAxisTitleAdjacentToAxis`
Títulos dos eixos	`msoElementPrimaryCategoryAxisTitleHorizontal`
Títulos dos eixos	`msoElementPrimaryCategoryAxisTitleVertical`
Títulos dos eixos	`msoElementPrimaryCategoryAxisTitleRotated`
Títulos dos eixos	`msoElementSecondaryCategoryAxisTitleAdjacentToAxis`
Títulos dos eixos	`msoElementSecondaryCategoryAxisTitleBelowAxis`
Títulos dos eixos	`msoElementSecondaryCategoryAxisTitleHorizontal`
Títulos dos eixos	`msoElementSecondaryCategoryAxisTitleNone`

Ícone da guia layout	Constante do elemento gráfico
Títulos dos eixos	msoElementSecondaryCategoryAxisTitleRotated
Títulos dos eixos	msoElementSecondaryCategoryAxisTitleVertical
Títulos dos eixos	msoElementPrimaryValueAxisTitleAdjacentToAxis
Títulos dos eixos	msoElementPrimaryValueAxisTitleBelowAxis
Títulos dos eixos	msoElementPrimaryValueAxisTitleHorizontal
Títulos dos eixos	msoElementPrimaryValueAxisTitleNone
Títulos dos eixos	msoElementPrimaryValueAxisTitleRotated
Legenda	msoElementLegendNone
Legenda	msoElementLegendRight
Legenda	msoElementLegendTop
Legenda	msoElementLegendLeft
Legenda	msoElementLegendBottom
Legenda	msoElementLegendRightOverlay
Legenda	msoElementLegendLeftOverlay
Rótulos de dados	msoElementDataLabelCenter
Rótulos de dados	msoElementDataLabelInsideEnd
Rótulos de dados	msoElementDataLabelNone
Rótulos de dados	msoElementDataLabelInsideBase
Rótulos de dados	msoElementDataLabelOutSideEnd
Rótulos de dados	msoElementDataLabelTop
Rótulos de dados	msoElementDataLabelBottom
Rótulos de dados	msoElementDataLabelRight
Rótulos de dados	msoElementDataLabelLeft
Rótulos de dados	msoElementDataLabelShow
Rótulos de dados	msoElementDataLabelBestFit
Tabela de dados	msoElementDataTableNone
Tabela de dados	msoElementDataTableShow
Tabela de dados	msoElementDataTableWithLegendKeys
Eixo	msoElementPrimaryCategoryAxisNone
Eixo	msoElementPrimaryCategoryAxisShow
Eixo	msoElementPrimaryCategoryAxisWithoutLabels
Eixo	msoElementPrimaryCategoryAxisReverse
Eixo	msoElementPrimaryCategoryAxisThousands
Eixo	msoElementPrimaryCategoryAxisMillions
Eixo	msoElementPrimaryCategoryAxisBillions
Eixo	msoElementPrimaryCategoryAxisLogScale
Eixo	msoElementSecondaryCategoryAxisNone
Eixo	msoElementSecondaryCategoryAxisShow

Tabela 11.2 Constantes disponíveis com SetElement (*cont.*)

Ícone da guia *layout*	Constante do elemento gráfico
Eixo	msoElementSecondaryCategoryAxisWithoutLabels
Eixo	msoElementSecondaryCategoryAxisReverse
Eixo	msoElementSecondaryCategoryAxisThousands
Eixo	msoElementSecondaryCategoryAxisMillions
Eixo	msoElementSecondaryCategoryAxisBillions
Eixo	msoElementSecondaryCategoryAxisLogScale
Eixo	msoElementPrimaryValueAxisNone
Eixo	msoElementPrimaryValueAxisShow
Eixo	msoElementPrimaryValueAxisThousands
Eixo	msoElementPrimaryValueAxisMillions
Eixo	msoElementPrimaryValueAxisBillions
Eixo	msoElementPrimaryValueAxisLogScale
Eixo	msoElementSecondaryValueAxisNone
Eixo	msoElementSecondaryValueAxisMillions
Eixo	msoElementSecondaryValueAxisBillions
Eixo	msoElementSecondaryValueAxisLogScale
Eixo	msoElementSeriesAxisNone
Eixo	msoElementSeriesAxisShow
Eixo	msoElementSeriesAxisReverse
Eixo	msoElementSeriesAxisWithoutLabeling
Linhas de grade	msoElementPrimaryCategoryGridLinesNone
Linhas de grade	msoElementPrimaryCategoryGridLinesMajor
Linhas de grade	msoElementPrimaryCategoryGridLinesMinor
Linhas de grade	msoElementPrimaryCategoryGridLinesMinorMajor
Linhas de grade	msoElementSecondaryCategoryGridLinesNone
Linhas de grade	msoElementSecondaryCategoryGridLinesMajor
Linhas de grade	msoElementSecondaryCategoryGridLinesMinor
Linhas de grade	msoElementSecondaryCategoryGridLinesMinorMajor
Linhas de grade	msoElementPrimaryValueGridLinesNone
Linhas de grade	msoElementPrimaryValueGridLinesMajor
Linhas de grade	msoElementPrimaryValueGridLinesMinor
Linhas de grade	msoElementPrimaryValueGridLinesMinorMajor
Linhas de grade	msoElementSecondaryValueGridLinesNone
Linhas de grade	msoElementSecondaryValueGridLinesMajor
Linhas de grade	msoElementSecondaryValueGridLinesMinor
Linhas de grade	msoElementSecondaryValueGridLinesMinorMajor
Linhas de grade	msoElementSeriesAxisGridLinesNone

Ícone da guia layout	Constante do elemento gráfico
Linhas de grade	msoElementSeriesAxisGridLinesMajor
Linhas de grade	msoElementSeriesAxisGridLinesMinor
Linhas de grade	msoElementSeriesAxisGridLinesMinorMajor
Área de plotagem	msoElementPlotAreaNone
Área de plotagem	msoElementPlotAreaShow
Parede do gráfico	msoElementChartWallNone
Parede do gráfico	msoElementChartWallShow
Base do gráfico	msoElementChartFloorNone
Base do gráfico	msoElementChartFloorShow
Linha de tendência	msoElementTrendlineNone
Linha de tendência	msoElementTrendlineAddLinear
Linha de tendência	msoElementTrendlineAddExponential
Linha de tendência	msoElementTrendlineAddLinearForecast
Linha de tendência	msoElementTrendlineAddTwoPeriodMovingAverage
Linhas	msoElementLineNone
Linhas	msoElementLineDropLine
Linhas	msoElementLineHiLoLine
Linhas	msoElementLineDropHiLoLine
Linhas	msoElementLineSeriesLine
Barras acima/abaixo	msoElementUpDownBarsNone
Barras acima/abaixo	msoElementUpDownBarsShow
Barra de erros	msoElementErrorBarNone
Barra de erros	msoElementErrorBarStandardError
Barra de erros	msoElementErrorBarPercentage
Barra de erros	msoElementErrorBarStandardDeviation

ATENÇÃO

Se tentar formatar um elemento ausente, o Excel retorna um erro –2147467259 Method Failed.

Alterando um título de gráfico com o VBA

Os menus predefinidos Layout permitem adicionar um título acima de um gráfico, mas não permitem mudar os caracteres no título do gráfico ou do eixo do gráfico.

Na interface com o usuário, você simplesmente dá um clique duplo no título do gráfico e digita um novo texto para alterá-lo. Infelizmente, o programa de gravação de macros não grava essa ação.

Para especificar um título de gráfico, você deve digitar esse código:

```
ActiveChart.ChartTitle.Caption = "My Chart"
```

De maneira semelhante, você pode especificar os títulos do eixo utilizando a propriedade `Caption`. O código a seguir muda o título do eixo no eixo de categoria:

```
ActiveChart.Axes(xlCategory, xlPrimary).AxisTitle.Caption = "Meses"
```

Emulando alterações na guia Formatar

No Excel 2007, o programa de gravação de macros não grava todas as ações que acontecem na guia Formatar ou nas caixas de diálogo na guia Layout. Isso é incrivelmente frustrante. É muito frustrante porque o Excel 2003 podia gravar essas alterações com o programa de gravação de macros. Uma solução para isso, se você ainda tem o Excel 2003 instalado, é formatar o gráfico no Excel 2003 enquanto o programa de gravação de macros estiver ativo. Depois você pode utilizar esse código no Excel 2007, embora não possa usar todos os recursos de formatação novos. Para informações sobre como detectar elementos gráficos sem a assistência do programa de gravação de macros, veja a seção "Utilizando a janela Inspeção para descobrir configurações de objeto", mais adiante neste capítulo.

NOVO Utilizando o método `Format` para acessar novas opções de formatação

O Excel 2007 introduz um novo objeto chamado objeto ChartFormat, que contém as configurações para `Fill`, `Glow`, `Line`, `PictureFormat`, `Shadow`, `SoftEdge`, `TextFrame2` e `ThreeD`. Você pode acessar o objeto ChartFormat utilizando o método `Format` em muitos elementos gráficos. A Tabela 11.3 relaciona uma amostragem de elementos gráficos que podem ser formatados com o método `Format`.

Tabela 11.3 Elementos gráficos aos quais a formatação se aplica

Elemento gráfico	VBA para referenciar esse elemento gráfico
Título do gráfico	`ChartTitle`
Título do eixo – Categorias	`Axes(xlCategory, xlPrimary).AxisTitle`
Título do eixo – Valor	`Axes(xlValue, xlPrimary).AxisTitle`
Legenda	`Legend`
Rótulos de dados para Série 1	`SeriesCollection(1).DataLabels`
Rótulos de dados para Ponto 2	`SeriesCollection(1).DataLabels(2)` ou `SeriesCollection(1).Points(2).DataLabel`
Tabela de dados	`DataTable`
Eixo – Horizontal	`Axes(xlCategory, xlPrimary)`
Eixo – Vertical	`Axes(xlValue, xlPrimary)`
Eixo – Série (gráficos de área apenas)	`Axes(xlSeries, xlPrimary)`
Linhas de grade principais	`Axes(xlValue, xlPrimary).MajorGridlines`
Linhas de grade secundárias	`Axes(xlValue, xlPrimary).MinorGridlines`
Área de plotagem	`PlotArea`
Área do gráfico	`ChartArea`
Parede do gráfico	`Walls`
Parede de fundo do gráfico	`BackWall`
Parede lateral do gráfico	`SideWall`
Base do gráfico	`Floor`
Linha de tendência para Série 1	`SeriesCollection(1).TrendLines(1)`
Linhas de projeção	`ChartGroups(1).DropLines`
Barras Acima/Abaixo	`ChartGroups(1).UpBars`
Barras de erro	`SeriesCollection(1).ErrorBars`
Série (1)	`SeriesCollection(1)`
Ponto de dados para Série 1	`SeriesCollection(1).Points(3)`

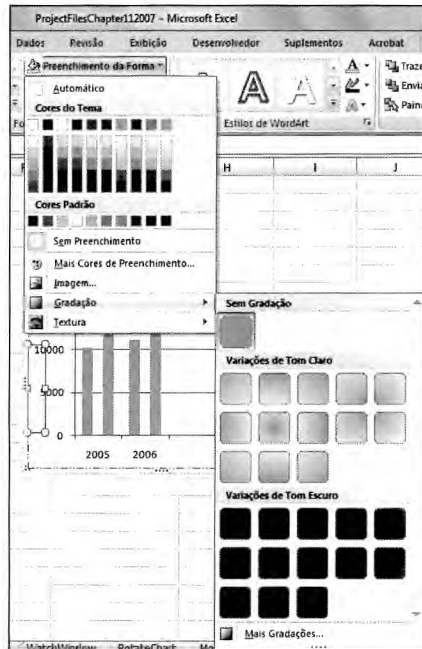
O método `Format` é o gateway para configurações como `Fill`, `Glow` e assim por diante. Cada um desses objetos tem opções diferentes. As seções a seguir dão exemplos de como configurar cada tipo de formato.

Mudando o preenchimento de um objeto

Como mostrado na Figura 11.6, a lista suspensa Preenchimento de Forma na guia Formatar permite escolher *uma única* cor, uma gradação, uma imagem ou uma textura para o preenchimento.

Para aplicar uma cor específica, você pode utilizar a configuração RGB (*red, green, blue*). Para criar uma cor, você especifica um valor de `0` a `255` para níveis de vermelho, verde e azul. O código a seguir aplica um preenchimento azul simples:

```
Dim cht As Chart
Dim upb As UpBars
Set cht = ActiveChart
Set upb = cht.ChartGroups(1).UpBars
upb.Format.Fill.ForeColor.RGB = RGB(0, 0, 255)
```

Figura 11.6
As opções de preenchimento são cor sólida, gradração, textura ou imagem.

Se quiser que um objeto tenha a cor específica do tom de um tema, utilize a propriedade `ObjectThemeColor`. O código a seguir altera a cor da barra da primeira série para acentuar a cor 6 (que é uma cor alaranjada no tema Office, mas poderia ser outra cor se a pasta de trabalho estivesse utilizando um tema diferente):

```
Sub ApplyThemeColor()
    Dim cht As Chart
    Dim ser As Series
    Set cht = ActiveChart
    Set ser = cht.SeriesCollection(1)
    ser.Format.Fill.ForeColor.ObjectThemeColor = msoThemeColorAccent6
End Sub
```

Para aplicar uma textura predefinida, utilize o método PresetTextured. O código a seguir aplica uma textura de mármore verde à segunda série, mas há 20 texturas diferentes disponíveis:

```
Sub ApplyTexture()
    Dim cht As Chart
    Dim ser As Series
    Set cht = ActiveChart
    Set ser = cht.SeriesCollection(2)
    ser.Format.Fill.PresetTextured (msoTextureGreenMarble)
End Sub
```

> **DICA** Quando você digita PresetTextured seguido por um parêntese aberto, o Editor do VB oferece uma lista completa de possíveis valores de textura.

Para preencher as barras de uma série de dados com uma imagem, utilizamos o método `UserPicture` e especificamos o caminho e o nome de arquivo de uma imagem no computador, como no seguinte exemplo:

```
Sub FormatWithPicture()
    Dim cht As Chart
    Dim ser As Series
    Set cht = ActiveChart
    Set ser = cht.SeriesCollection(1)
    MyPic = "C:\PodCastTitle1.jpg"
    ser.Format.Fill.UserPicture (MyPic)
End Sub
```

As gradações são mais difíceis de serem especificadas que os preenchimentos. O Excel 2007 oferece três métodos que ajudam a configurar as gradações comuns. Os métodos `OneColorGradient` e `TwoColorGradient` requerem que se especifique uma direção de gradação, como `msoGradientFromCorner`. Você pode então especificar um de quatro estilos, numerados de 1 a 4, dependendo se quer que a gradação inicie na parte superior esquerda, superior direita, inferior esquerda ou direita. Depois de utilizar um método de gradação, você precisa especificar as configurações `ForeColor` e `BackColor` do objeto. O macro a seguir define uma gradação de duas cores usando duas cores de tema:

```
Sub TwoColorGradient()
    Dim cht As Chart
    Dim ser As Series
    Set cht = ActiveChart
    Set ser = cht.SeriesCollection(1)
    MyPic = "C:\PodCastTitle1.jpg"
    ser.Format.Fill.TwoColorGradient msoGradientFromCorner, 3
    ser.Format.Fill.ForeColor.ObjectThemeColor = msoThemeColorAccent6
    ser.Format.Fill.BackColor.ObjectThemeColor = msoThemeColorAccent2
End Sub
```

Ao utilizar o método `OneColorGradient`, você especifica uma direção, um estilo (1 a 4) e um valor de escuridão entre 0 e 1 (0 para gradações mais escuras ou 1 para as mais claras).

Ao utilizar o método `PresetGradient`, você especifica uma direção, um estilo (`1 a 4`) e o tipo de gradação (por exemplo, `msoGradientBrass`, `msoGradientLateSunset` ou `msoGradientRainbow`). Novamente, enquanto você digita esse código no Editor do VB, a ferramenta AutoCompletar fornece uma lista completa dos tipos de gradações predefinidas disponíveis.

Formatando configurações de linha

O objeto LineFormat formata uma linha ou a borda de um objeto. Numerosas propriedades de uma linha podem ser modificadas, como cor, setas, estilo de traço e assim por diante.

A macro a seguir formata a linha de tendência da primeira série em um gráfico:

```
Sub FormatLineOrBorders()
    Dim cht As Chart
    Set cht = ActiveChart
    With cht.SeriesCollection(1).Trendlines(1).Format.Line
        .DashStyle = msoLineLongDashDotDot
        .ForeColor.RGB = RGB(50, 0, 128)
        .BeginArrowheadLength = msoArrowheadShort
        .BeginArrowheadStyle = msoArrowheadOval
        .BeginArrowheadWidth = msoArrowheadNarrow
        .EndArrowheadLength = msoArrowheadLong
        .EndArrowheadStyle = msoArrowheadTriangle
        .EndArrowheadWidth = msoArrowheadWide
    End With
End Sub
```

Ao formatar uma borda, as configurações de seta não são relevantes; portanto, o código é mais curto que aquele para formatar uma linha. O macro a seguir formata a borda de um gráfico:

```
Sub FormatBorder()
    Dim cht As Chart
    Set cht = ActiveChart
    With cht.ChartArea.Format.Line
        .DashStyle = msoLineLongDashDotDot
        .ForeColor.RGB = RGB(50, 0, 128)
    End With
End Sub
```

Formatando configurações de brilho

Para criar um brilho, você têm de especificar uma cor e um raio. O valor de raio pode ser de `1` a `20`. Um raio de valor `1` é raramente visível e um raio de `20` é muito espesso.

> NOTA
> Um brilho só pode ser aplicado ao contorno de forma. Se você tentar adicionar brilho a um objeto cujo contorno é definido como `None`, o brilho não poderá ser visto.

O macro a seguir adiciona uma linha em torno do título e adiciona um brilho em torno dessa linha:

```
Sub AddGlowToTitle()
    Dim cht As Chart
    Set cht = ActiveChart
    cht.ChartTitle.Format.Line.ForeColor.RGB = RGB(255, 255, 255)
    cht.ChartTitle.Format.Line.DashStyle = msoLineSolid
    cht.ChartTitle.Format.Glow.Color.ObjectThemeColor = msoThemeColorAccent6
    cht.ChartTitle.Format.Glow.Radius = 8
End Sub
```

Formatando configurações de sombra

Uma sombra é composta de uma cor, uma transparência e vários pontos pelos quais a sombra deve ser deslocada do objeto. Se você aumentar o número de pontos, parecerá que o objeto está mais distante da superfície do gráfico. O deslocamento horizontal é conhecido como `OffsetX` e o deslocamento vertical é conhecido como `OffsetY`.

A macro a seguir adiciona uma sombra azul leve à caixa que cerca uma legenda:

```
Sub FormatShadow()
    Dim cht As Chart
    Set cht = ActiveChart
    With cht.Legend.Format.Shadow
        .ForeColor.RGB = RGB(0, 0, 128)
        .OffsetX = 5
        .OffsetY = -3
        .Transparency = 0.5
        .Visible = True
    End With
End Sub
```

Formatando configurações de reflexo

Não é possível aplicar reflexos a elementos gráficos. As configurações Reflexo na guia Formatar estão constantemente desativadas quando um gráfico está selecionado. De maneira semelhante, o objeto ChartFormat não tem um objeto reflexo.

Formatando bordas suaves

Há seis níveis de configurações de bordas suaves. As configurações suavizam as bordas em 1, 2.5, 5, 10, 25 ou 50 pontos. A primeira configuração está pouco visível. Em geral, as mais altas são maiores que a maioria dos elementos gráficos que você provavelmente formatará.

A Microsoft diz que a sintaxe a seguir é a adequada para `Borda Suave`:

```
Chart.Seriess(1).Points(i).Format.SoftEdge.Type = msoSoftEdgeType1
```

Entretanto, `msoSoftEdgeType1` e palavras como esta são de fato variáveis definidas pelo Excel. Para tentar um truque interessante, vá ao Editor do VB e abra a janela Verificação Imediata pressionando Ctrl+G. Nessa janela, digite **`Print msoSoftEdgeType2`** e pressione Enter. A janela Verificação Imediata informará que utilizar essa palavra é equivalente a digitar 2. Então, você poderá utilizar `msoSoftEdgeType2` ou o valor 2.

Se você utilizar `msoSoftEdgeType2`, seu código será um pouco mais fácil de entender do que se utilizar simplesmente `2`. Mas se você espera formatar cada ponto de uma série de dados com um formato diferente, talvez queira utilizar um loop como este, caso em que é muito mais fácil utilizar apenas os números de `1` a `6` do que `msoSoftEdgeType1` a `msoSoftEdgeType6`, conforme mostrado nesta macro:

```
Sub FormatSoftEdgesWithLoop()
    Dim cht As Chart
    Dim ser As Series
```

```
    Set cht = ActiveChart
    Set ser = cht.SeriesCollection(1)
    For i = 1 To 6
        ser.Points(i).Format.SoftEdge.Type = i
    Next i
End Sub
```

ATENÇÃO

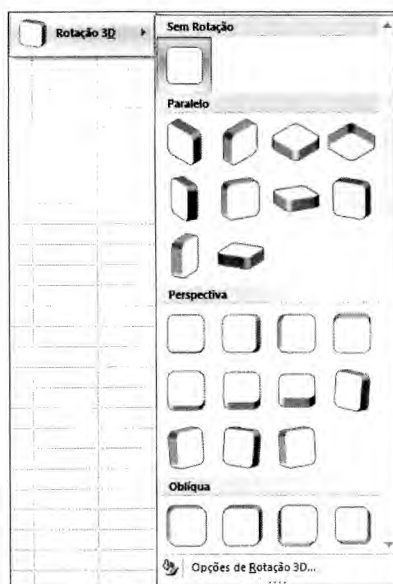
É um pouco estranho que bordas suaves sejam definidas como um número fixo de pontos. Em um gráfico dimensionado para se ajustar a uma folha inteira de papel, uma borda suave de 10 pontos funciona bem. Mas, se você redimensionar o gráfico para que caibam seis gráficos em uma página, aplicar uma borda suave de 10 pontos a todos os lados de uma coluna pode fazê-la desaparecer completamente.

Formatando configurações de rotação 3-D

As configurações 3-D tratam três menus diferentes na guia Formatar. Na lista suspensa Efeitos de Forma, as configurações sob Predefinição, Bisel e Rotação 3-D são realmente tratadas pelo objeto ThreeD no objeto ChartFormat. Esta seção discute configurações que afetam a rotação 3-D. A próxima seção discute as configurações que afetam os formatos bisel e 3-D.

Os métodos e propriedades que podem ser definidos para o objeto ThreeD são muito amplos. De fato, as configurações 3-D no VBA incluem opções mais predefinidas do que os menus da guia Formatar.

A Figura 11.7 mostra as predefinições disponíveis no menu flutuante Rotação 3-D.

Figura 11.7
Enquanto o menu Rotação 3-D tem 25 predefinições, o VBA oferece 62.

Para aplicar uma das predefinições de rotação 3-D a um elemento gráfico, usamos o método `SetPresetCamera`, como mostrado aqui:

```
Sub Assign3DPreset()
    Dim cht As Chart
    Dim shp As Shape
    Set cht = ActiveChart
    Set shp = cht.Shapes(1)
    shp.ThreeD.SetPresetCamera msoCameraIsometricLeftDown
End Sub
```

A Tabela 11.4 relaciona todos os possíveis valores `SetPresetCamera`. Se a primeira coluna indicar que é um bônus ou um estilo do Excel 2003, o valor será uma predefinição que está disponível no VBA, mas não incluída pela Microsoft no menu flutuante Rotação 3-D.

Tabela 11.4 Formatos predefinidos 3-D e seus valores constantes do VBA

Localização do menu	Descrição	Valor do VBA
Grupo Paralelo, linha 1, coluna 1	Isométrica para Esquerda e para Baixo	`msoCameraIsometricLeftDown`
Grupo Paralelo, linha 1, coluna 2	Isométrica para Direita e para Cima	`msoCameraIsometricRightUp`

Localização do menu	Descrição	Valor do VBA
Grupo Paralelo, linha 1, coluna 3	Isométrica para Cima	msoCameraIsometricTopUp
Grupo Paralelo, linha 1, coluna 4	Isométrica para Baixo	msoCameraIsometricBottomDown
Grupo Paralelo, linha 2, coluna 1	Isométrica Fora do Eixo 1 à Esquerda	msoCameraIsometricOffAxis1Left
Grupo Paralelo, linha 2, coluna 2	Isométrica Fora do Eixo 1 à Direita	msoCameraIsometricOffAxis1Right
Grupo Paralelo, linha 2, coluna 3	Isométrica Fora do Eixo 1 Acima	msoCameraIsometricOffAxis1Top
Grupo Paralelo, linha 2, coluna 4	Isométrica Fora do Eixo 2 à Esquerda	msoCameraIsometricOffAxis2Left
Grupo Paralelo, linha 3, coluna 1	Isométrica Fora do Eixo 2 à Direita	msoCameraIsometricOffAxis2Right
Grupo Paralelo, linha 3, coluna 2	Isométrica Fora do Eixo 2 Acima	msoCameraIsometricOffAxis2Top
Grupo Paralelo, seleção de bônus	Isométrica para Baixo e para Cima	msoCameraIsometricBottomUp
Grupo Paralelo, seleção de bônus	Isométrica para Esquerda	msoCameraIsometricLeftUp
Grupo Paralelo, seleção de bônus	Isométrica Fora do Eixo 3 para Baixo	msoCameraIsometricOffAxis3Bottom
Grupo Paralelo, seleção de bônus	Isométrica Fora do Eixo 3 à Esquerda	msoCameraIsometricOffAxis3Left
Grupo Paralelo, seleção de bônus	Isométrica Fora do Eixo 3 à Direita	msoCameraIsometricOffAxis3Right
Grupo Paralelo, seleção de bônus	Isométrica Fora do Eixo 4 para Baixo	msoCameraIsometricOffAxis4Bottom
Grupo Paralelo, seleção de bônus	Isométrica Fora do Eixo 4 à Esquerda	msoCameraIsometricOffAxis4Left
Grupo Paralelo, seleção de bônus	Isométrica Fora do Eixo 4 à Direita	msoCameraIsometricOffAxis4Right
Grupo Paralelo, seleção de bônus	Isométrica para Direita e para Baixo	msoCameraIsometricRightDown
Grupo Paralelo, seleção de bônus	Isométrica para Baixo	msoCameraIsometricTopDown
Grupo Perspectiva, linha 1, coluna 1	Perspectiva Frontal	msoCameraPerspectiveFront
Grupo Perspectiva, linha 1, coluna 2	Perspectiva à Esquerda	msoCameraPerspectiveLeft
Grupo Perspectiva, linha 1, coluna 3	Perspectiva à Direita	msoCameraPerspectiveRight
Grupo Perspectiva, linha 1, coluna 4	Perspectiva Abaixo	msoCameraPerspectiveBelow
Grupo Perspectiva, linha 2, coluna 1	Perspectiva Acima	msoCameraPerspectiveAbove
Grupo Perspectiva, linha 2, coluna 2	Perspectiva Moderadamente Relaxada	msoCameraPerspectiveRelaxedModerately
Grupo Perspectiva, linha 2, coluna 3	Perspectiva Relaxada	msoCameraPerspectiveRelaxed
Grupo Perspectiva, linha 2, coluna 4	Perspectiva Contrastante à Esquerda	msoCameraPerspectiveContrastingLeftFacing
Grupo Perspectiva, linha 3, coluna 1	Perspectiva Contrastante à Direita	msoCameraPerspectiveContrastingRightFacing
Grupo Perspectiva, linha 3, coluna 2	Perspectiva Heróica à Extrema Esquerda	msoCameraPerspectiveHeroicExtremeLeftFacing
Grupo Perspectiva, linha 3, coluna 3	Perspectiva Heróica à Extrema Direita	msoCameraPerspectiveHeroic ExtremeRightFacing
Grupo Perspectiva, seleção de bônus	Perspectiva Acima e à Esquerda	msoCameraPerspectiveAboveLeftFacing
Grupo Perspectiva, seleção de bônus	Perspectiva Acima e à Direita	msoCameraPerspectiveAboveRightFacing
Grupo Perspectiva, seleção de bônus	Perspectiva Heróica à Esquerda	msoCameraPerspectiveHeroicLeftFacing
Grupo Perspectiva, seleção de bônus	Perspectiva Heróica à Direita	msoCameraPerspectiveHeroicRightFacing
Grupo Perspectiva, estilos do Excel 2003	Perspectiva de Legado para Baixo	msoCameraLegacyPerspectiveBottom
Grupo Perspectiva, estilos do Excel 2003	Perspectiva de Legado para Baixo e para Esquerda	msoCameraLegacyPerspectiveBottomLeft
Grupo Perspectiva, estilos do Excel 2003	Perspectiva de Legado para Baixo e para Direita	msoCameraLegacyPerspectiveBottomRight

Tabela 11.4 Formatos predefinidos 3-D e seus valores constantes do VBA (*cont.*)

Localização do menu	Descrição	Valor do VBA
Grupo Perspectiva, estilos do Excel 2003	Perspectiva de Legado Frontal	`msoCameraLegacyPerspectiveFront`
Grupo Perspectiva, estilos do Excel 2003	Perspectiva de Legado à Esquerda	`msoCameraLegacyPerspectiveLeft`
Grupo Perspectiva, estilos do Excel 2003	Perspectiva de legado à Direita	`msoCameraLegacyPerspectiveRight`
Grupo Perspectiva, estilos do Excel 2003	Perspectiva de Legado para Cima	`msoCameraLegacyPerspectiveTop`
Grupo Perspectiva, estilos do Excel 2003	Perspectiva de Legado para Cima e para Esquerda	`msoCameraLegacyPerspectiveTopLeft`
Grupo Perspectiva, estilos do Excel 2003	Perspectiva de Legado para Cima e para Direita	`msoCameraLegacyPerspectiveTopRight`
Grupo Oblíqua, linha 1, coluna 1	Oblíqua para Cima e para Esquerda	`msoCameraObliqueTopLeft`
Grupo Oblíqua, linha 1, coluna 2	Oblíqua para Cima e para Direita	`msoCameraObliqueTopRight`
Grupo Oblíqua, linha 1, coluna 3	Oblíqua para Baixo e para Esquerda	`msoCameraObliqueBottomLeft`
Grupo Oblíqua, linha 1, coluna 4	Oblíqua para Baixo e para Direita	`msoCameraObliqueBottomRight`
Grupo Oblíqua, seleção de bônus	Oblíqua para Baixo	`msoCameraObliqueBottom`
Grupo Oblíqua, seleção de bônus	Oblíqua à Esquerda	`msoCameraObliqueLeft`
Grupo Oblíqua, seleção de bônus	Oblíqua à Direita	`msoCameraObliqueRight`
Grupo Oblíqua, seleção de bônus	Oblíqua Acima	`msoCameraObliqueTop`
Grupo Oblíqua, seleção de bônus	Frente Ortográfica	`msoCameraOrthographicFront`
Grupo Oblíqua, estilos do Excel 2003	Oblíqua de legado para Baixo	`msoCameraLegacyObliqueBottom`
Grupo Oblíqua, estilos do Excel 2003	Oblíqua de Legado para Esquerda Inferior	`msoCameraLegacyObliqueBottomLeft`
Grupo Oblíqua, estilos do Excel 2003	Oblíqua de Legado para Baixo e para Direita	`msoCameraLegacyObliqueBottomRight`
Grupo Oblíqua, estilos do Excel 2003	Oblíqua de Legado Frontal	`msoCameraLegacyObliqueFront`
Grupo Oblíqua, estilos do Excel 2003	Oblíqua de Legado para Esquerda	`msoCameraLegacyObliqueLeft`
Grupo Oblíqua, estilos do Excel 2003	Oblíqua de Legado para Direita	`msoCameraLegacyObliqueRight`
Grupo Oblíqua, estilos do Excel 2003	Oblíqua de Legado Acima	`msoCameraLegacyObliqueTop`
Grupo Oblíqua, estilos do Excel 2003	Oblíqua de Legado para Cima e para a Esquerda	`msoCameraLegacyObliqueTopLeft`
Grupo Oblíqua, estilos do Excel 2003	Oblíqua de Legado para Cima e para Direita	`msoCameraLegacyObliqueTopRight`

Se preferir não usar as predefinições, você pode controlar explicitamente a rotação em torno do eixo x, y ou z. Você pode usar as seguintes propriedades e métodos para alterar a rotação de um objeto:

- **`RotationX`** — Retorna ou configura a rotação da forma extruída em torno do eixo x, em graus. Isso pode ser um valor de `-90` a `90`. Um valor positivo indica rotação para cima; um valor negativo indica rotação para baixo.
- **`RotationY`** — Retorna ou configura a rotação da forma extruída em torno do eixo y, em graus. Pode ser um valor de `-90` a `90`. Um valor positivo indica rotação para a esquerda; um valor negativo indica rotação para a direita.
- **`RotationZ`** — Retorna ou configura a rotação da forma extruída em torno do eixo `z`, em graus. Pode ser um valor de `-90` a `90`. Um valor positivo indica rotação para cima; um valor negativo indica rotação para baixo.
- **`IncrementRotationX`** — Altera a rotação da forma especificada em torno do eixo x pelo número especificado de graus. Determina um incremento de `-90` a `90`. Os graus negativos inclinam o objeto para baixo e os positivos, para cima. (Você pode utilizar a propriedade `RotationX` para configurar a rotação absoluta da forma em torno do eixo x.)

- `IncrementRotationY` — Altera a rotação da forma especificada em torno do eixo y pelo número especificado de graus. Um valor positivo inclina o objeto para esquerda e um valor negativo, para a direita. (Você pode utilizar a propriedade `RotationY` para configurar a rotação absoluta da forma em torno do eixo y.)
- `IncrementRotationZ` — Altera a rotação da forma especificada em torno do eixo x pelo número especificado de graus. Um valor positivo inclina o objeto para esquerda e um valor negativo inclina-o para a direita. (Você pode utilizar a propriedade `RotationZ` para definir a rotação absoluta da forma em torno do eixo z.)
- `IncrementRotationHorizontal` — Altera a rotação da forma especificada horizontalmente pelo número especificado de graus. Determina um incremento de `-90` a `90` para especificar quanto (em graus) a rotação da forma deve ser alterada horizontalmente. Um valor positivo move a forma para a esquerda; um valor negativo move-o para a direta.
- `IncrementRotationVertical` — Altera a rotação da forma especificada verticalmente pelo número especificado de graus. Especifica um incremento de `-90` a `90` para especificar quanto (em graus) a rotação da forma deve ser alterada verticalmente. Um valor positivo move a forma para a esquerda; um valor negativo move-a para a direta.
- `ResetRotation` — Redefine a rotação de extrusão em torno do eixo x e y para `0`, de modo que a frente da extrusão fique de frente. Esse método não redefine a rotação em torno do eixo x.

Alterando o bisel e o formato 3-D

Há 12 predefinições no menu flutuante Bisel. Essas predefinições afetam o bisel (ou chanfro) na parte superior do objeto. Normalmente em gráficos você vê a face superior; entretanto, há algumas rotações esquisitas de um gráfico 3-D onde se vê a face da parte inferior de gráficos e elementos.

A caixa de diálogo Formatar Forma contém as mesmas 12 predefinições do menu flutuante Bisel, mas permite aplicar a predefinição às faces superior ou inferior. Você também pode controlar a largura e a altura do bisel. As propriedades e métodos do VBA correspondem às configurações na categoria Formato 3D da caixa de diálogo Definir forma (veja Figura 11.8).

Você configura o tipo de bisel utilizando as propriedades `BevelTopType` e `BevelBottomType`. Você pode modificar mais ainda o tipo de bisel, configurando o valor `BevelTopInset` para definir a largura e o valor `BevelTopDepth` para definir a altura. A macro a seguir adiciona um bisel às colunas da Série 1:

```
Sub AssignBevel()
    Dim cht As Chart
    Dim ser As Series
    Set cht = ActiveChart
    Set ser = cht.SeriesCollection(1)
    ser.Format.ThreeD.Visible = True
    ser.Format.ThreeD.BevelTopType = msoBevelCircle
    ser.Format.ThreeD.BevelTopInset = 16
    ser.Format.ThreeD.BevelTopDepth = 6
End Sub
```

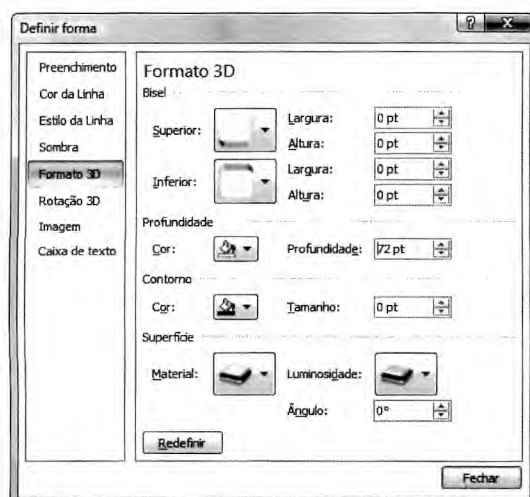
Figura 11.8
Você pode controlar as configurações de Formato 3D, como bisel, superfície e luminosidade.

As 12 possíveis configurações para o tipo de bisel são mostradas na Tabela 11.5; essas configurações correspondem às miniaturas no menu flutuante. Para desativar o bisel, utilize `msoBevelNone`.

Tabela 11.5 Tipos de bisel

Constante	Constante
msoBevelCircle	msoBevelConvex
msoBevelRelaxedInset	msoBevelSlope
msoBevelCross	msoBevelDivot
msoBevelCoolSlant	msoBevelRiblet
msoBevelAngle	msoBevelHardEdge
msoBevelSoftRound	msoBevelArtDeco

Normalmente, o tom da cor utilizado em um bisel é baseado na cor utilizada para preencher o objeto. Mas, se você quiser controlar a cor de extrusão, terá primeiro de especificar qual o tipo de cor de extrusão é personalizado e, depois, especificar um tom de cor de tema ou uma cor RGB, como no seguinte exemplo:

```
ser.Format.ThreeD.ExtrusionColorType = msoExtrusionColorCustom
'utilize isto:
ser.Format.ThreeD.ExtrusionColor.ObjectThemeColor = msoThemeColorAccent1
'ou isto:
ser.Format.ThreeD.ExtrusionColor.RGB = RGB(255, 0, 0)
```

Utilizamos a propriedade `Depth` para controlar a quantidade de extrusão no bisel e especificar a profundidade em pontos. Eis um exemplo:

```
ser.Format.ThreeD.Depth = 5
```

Para o contorno, você pode especificar a cor e o tamanho do contorno ou ambos. Podemos determinar a cor como um valor RGB ou uma cor de tema. Especificamos o tamanho em pontos, utilizando a propriedade `ContourWidth`. Eis um exemplo:

```
ser.Format.ThreeD.ContourColor.RGB = RGB(0, 255, 0)
ser.Format.ThreeD.ContourWidth = 10
```

As listas suspensas Superfície são controladas pelas seguintes propriedades:

- **`PresetMaterial`** — Contém opções da lista suspensa Material.
- **`PresetLighting`** — Contém escolhas da lista suspensa Luminosidade.
- **`LightAngle`** — Controla o ângulo a partir do qual a luz está refletindo no objeto.

O menu suspenso Material da categoria 3-D da caixa de diálogo Formato oferece 11 configurações, embora pareça que a Microsoft tenha projetado uma décima segunda configuração no modelo de objeto. Não está claro por que a Microsoft não oferece o estilo `SoftMetal` na caixa de diálogo, mas você pode utilizá-lo no VBA. Também há três estilos de legado no modelo de objeto que não estão disponíveis na caixa de diálogo Formato. Na teoria, o novo material `Plastic2` é melhor do que o antigo material `Plastic`. A Tabela 11.6 mostra as configurações de cada miniatura.

Tabela 11.6 Constantes do VBA para tipos de materiais

Tipo	Constantes dos VBA	Valor
Fosco	msoMaterialMatte2	5
Mate Quente	msoMaterialWarmMatte	8
Plástico	msoMaterialPlastic2	6
Metal	msoMaterialMetal2	7

	Tipo	Constantes dos VBA	Valor
	Borda Escura	msoMaterialDarkEdge	11
	Borda Suave	msoMaterialSoftEdge	12
	Plano	msoMaterialFlat	14
	Delineado	msoMaterialWireFrame	4
	Pó	msoMaterialPowder	10
	Pó Translúcido	msoMaterialTranslucentPowder	9
	Claro	msoMaterialClear	13
		msoMaterialMatte	1
		msoMaterialPlastic	2
		msoMaterialMetal	3
Bônus		msoMaterialSoftMetal	15

No Excel 2003, a propriedade do material era limitada a mate, metal, plástico e delineado. Aparentemente, a Microsoft não estava feliz com as antigas configurações de mate, metal e plástico. Ela continuou com esses valores para suportar gráficos legados, mas criou as novas configurações `Mate2`, `Plástico2` e `Metal2`. Essas configurações estão realmente disponíveis na caixa de diálogo. No VBA, você está livre para utilizar as configurações antigas ou novas. As colunas na Figura 11.9 comparam as configurações novas e antigas. A coluna final é para a configuração `MetalLeve` que a Microsoft deixou de fora da caixa de diálogo Formato. Provavelmente, essa foi uma decisão estética em vez de uma preocupação com a possibilidade de o computador travar. Você pode sentir-se à vontade para utilizar `msoMaterialSoftMetal` para criar uma aparência que tenha uma diferença sutil de gráficos que outras pessoas criam utilizando as configurações na caixa de diálogo Formato.

O menu suspenso Luminosidade da categoria 3-D da caixa de diálogo Formato oferece 15 configurações. O modelo de objeto oferece essas 15 configurações, mais 13 configurações de legado na barra de ferramentas Luminosidade do Excel 2003. A Tabela 11.7 mostra as configurações de cada uma dessas miniaturas.

Figura 11.9
Comparação de algumas predefinições de material novas e antigas.

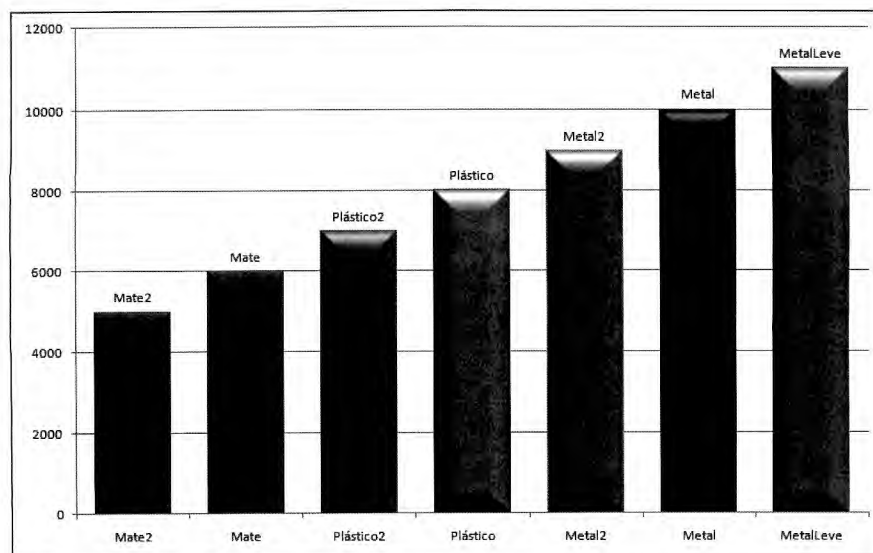



Tabela 11.7 Constantes do VBA para tipos de luminosidade

	Tipo	Constantes dos VBA	Valor
Categoria Neutro			
	Três Pontos	msoLightRigThreePoint	13
	Equilibrada	msoLightRigBalanced	14
	Suave	msoLightRigSoft	15
	Irregular	msoLightRigHarsh	16
	Intensa	msoLightRigFlood	17
	Contrastante	msoLightRigContrasting	18
Categoria Quente			
	De manhã	msoLightRigMorning	19
	Nascer do Sol	msoLightRigSunrise	20
	Pôr-do-sol	msoLightRigSunset	21
Categoria Elegante			
	Fria	msoLightRigChilly	22
	Congelada	msoLightRigFreezing	23
Categoria Especial			
	Plano	msoLightRigFlat	24
	Dois Pontos	msoLightRigTwoPoint	25
	Brilho	msoLightRigGlow	26
	Translúcida	msoLightRigBrightRoom	27
Categoria de legado			
	Plano 1	msoLightRigLegacyFlat1	1
	Plano 2	msoLightRigLegacyFlat2	2
	Plano 3	msoLightRigLegacyFlat3	3
	Plano 4	msoLightRigLegacyFlat4	4
	Irregular 1	msoLightRigLegacyHarsh1	9
	Irregular 2	msoLightRigLegacyHarsh2	10
	Irregular 3	msoLightRigLegacyHarsh3	11
	Irregular 4	msoLightRigLegacyHarsh4	12
	Normal 1	msoLightRigLegacyNormal1	5
	Normal 2	msoLightRigLegacyNormal2	6

Tipo	Constantes dos VBA	Valor
Normal 3	msoLightRigLegacyNormal3	7
Normal 4	msoLightRigLegacyNormal4	8
Misto	msoLightRigMixed	-2

Utilizando a janela Inspeção de Variáveis para descobrir configurações de objeto

É frustrante que o programa de gravação de macros não grave certas ações enquanto trabalhamos com gráficos. Na verdade, há duas frustrações. A primeira é que o programa de gravação de macros não grava a ação de criar elementos gráficos SmartArt, porque a Microsoft tomou uma decisão consciente para não permitir que você crie SmartArt utilizando o VBA. Não concordo com essa decisão, mas posso entender por que o programa de gravação de macros não grava esses passos.

A segunda é que, quando utilizamos a guia Formatar, o programa de gravação de macros não faz nada; mas, utilizando o VBA, é possível controlar todas as ações com a guia Formatar.

Com as versões anteriores do Excel, eu contava com o programa de gravação de macros para me ensinar que objetos, propriedades e métodos respondiam às várias ações na interface do Excel. Sem ele, torna-se muito difícil aprender esses aspectos.

Caso você precise utilizar uma propriedade que não tenha sido discutida neste livro, há uma maneira de explorar as propriedades de certos elementos gráficos. A seguir, um exemplo em que a macro define uma variável de objeto Chart e uma variável de objeto ChartGroup e então pára:

```
Sub ExploreChartElements()
    Dim cht As Chart
    Dim chtg As ChartGroup
    Dim ser As Series
    Set cht = ActiveChart
    Set chtg = cht.ChartGroups(1)
    Set ser = cht.SeriesCollection(1)
    Stop

End Sub
```

O comando `Stop` na macro é essencial para o sucesso dessa técnica. O Excel entra no modo Interrupção quando encontra o código `Stop`. Isso permite examinar as variáveis de objeto enquanto elas ainda estão no escopo.

Estes são os passos para descobrir novas propriedades de gráfico:

1. Insira a macro anterior em sua pasta de trabalho.
2. Crie um gráfico.
3. Selecione o gráfico.
4. Execute a macro. O VBA pára e destaca a linha `Stop` em amarelo. Você está agora no modo Interrupção.
5. Clique com o botão direito do mouse na variável de objeto `ser` e escolha Adicionar Inspeção de Variáveis. Clique em OK na caixa de diálogo Adicionar Inspeção de Variáveis. O Excel exibe uma nova janela Inspeção na parte inferior do Editor do VB. Essa janela exibe uma única linha com um par de óculos, um sinal de adição e o nome da variável, como mostrado na Figura 11.10.
6. Clique no sinal de adição ao lado da inspeção. Uma lista de muitas propriedades para a série se abre. Uma dessas propriedades é `Format`. É aqui que todas as configurações da guia Formatar são armazenadas.
7. Clique no sinal de adição ao lado da entrada `Format`. Ele expande para mostrar as configurações para `Fill`, `Glow`, `Line` e assim por diante.
8. Clique no sinal de adição ao lado da entrada `Fill`. Você vê muitas configurações que definem o preenchimento utilizado na Série 1. A configuração `GradientDegree` é destacada na Figura 11.11. Você pode ver que `Gradient Degree` é uma propriedade da propriedade `Fill` e que `Fill` é uma propriedade da propriedade `Format`. A partir disso, você pode determinar que o código adequado seria este:

```
ser.Format.Fill.GradientDegree = 0.8825
```

Figura 11.10
Inicialmente, a variável inspecionada mostra uma única linha inútil.

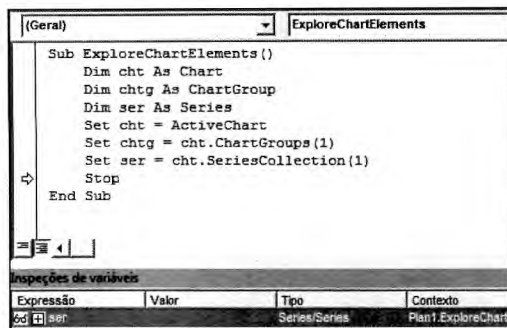
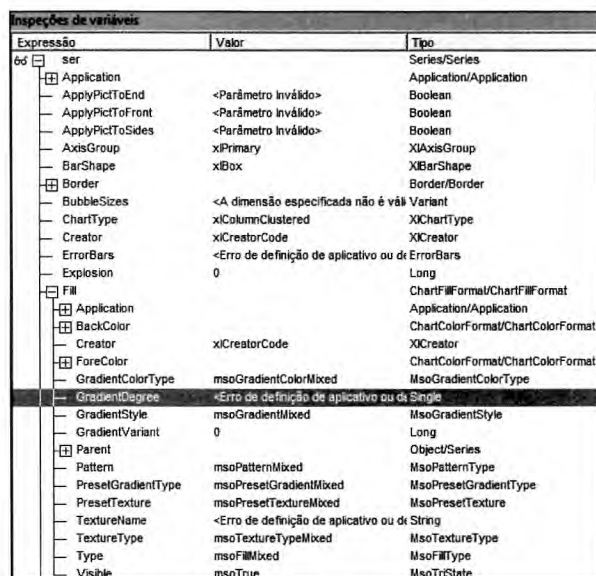



Figura 11.11
Depois de navegar pela janela Inspeção, você pode localizar uma propriedade sem o programa de gravação de macros.

Isso não é tão fácil quanto utilizar o programa de gravação de macros para examinar objetos, propriedades e métodos, mas possibilita descobrir como se escreve o código.

> **NOTA** Na macro `ExploreChartElements`, há variáveis para o gráfico, a série e o grupo de gráfico. Talvez você tenha de adicionar inspeções para cada uma dessas variáveis e começar a explorar para encontrar a configuração real.

A janela Inspeção é 'um pouco' viva. Com alguns passos, você pode alterar a formatação do gráfico na interface com o usuário do Excel e, então, retornar ao Editor do VB para descobrir as novas configurações:

1. Enquanto estiver no modo de Interrupção, volte para o Excel utilizando para Alt+Tab ou clicando no Excel na barra de tarefas.
2. Faça algumas alterações no gráfico ativo no Excel. Certifique-se de não desativar o gráfico.
3. Volte para o Editor do VB.
4. Há uma seta amarela à esquerda da linha `Stop` em sua janela Código. Arraste essa seta para cima, para apontar para a linha que define a variável inspecionada. No exemplo atual, você está inspecionando a variável `ser`; então, só precisa mover-se uma linha para cima, para a linha `Set ser`.
5. Pressione a tecla F8 para executar novamente a linha destacada em amarelo. A janela Inspeção é atualizada para mostrar as configurações definidas no passo 2.

Quando você tiver concluído a exploração, clique no botão Redefinir na barra de ferramentas do VBA (o ponto quadrado abaixo do menu Executar).

Estudo de caso

Utilizando a janela Inspeção para aprender configurações de rotação

Há um ícone na guia Layout que não é gravado pelo programa de gravação de macros. Se alterar a rotação de um gráfico 3-D, o programa de gravação de macros não gravará nada. Para ver um exemplo disso, você pode seguir estes passos:

1. Ative o programa de gravação de macros.
2. Selecione um gráfico de superfície.
3. Altere as configurações de rotação.
4. Pare o programa de gravação de macros.
5. Volte para o Excel e mude a formatação das colunas na primeira série. O Excel gravará essa macro, que é de pouca utilidade:

```
Sub RotateChartRecordedMacro()
'
'Macro RotateChartRecordedMacro
'
'
    ActiveSheet.ChartObjects("Chart 1").Activate

End Sub
```

Para resolver esse problema e aprender a girar programaticamente um gráfico, você pode seguir estes passos:

1. Digite essa macro no Editor do VB:

```
Sub ExploreChartElements()
    Dim cht As Chart
    Set cht = ActiveChart
    Stop
End Sub
```

2. No Excel, selecione o gráfico. Na guia Layout, escolha o ícone Rotação 3-D. Observe as configurações X, Y e de perspectiva (por exemplo, 200, 10 e 15).
3. Clique em Fechar para fechar a caixa de diálogo Formatar Área do Gráfico.
4. Volte para o VBA.
5. Execute a macro ExploreChartElements. VBA entra no modo de Interrupção.
6. Clique com o botão direito do mouse na variável `cht` e escolha Adicionar Inspeção de Variáveis. Clique em OK na caixa de diálogo Adicionar Inspeção de Variáveis.
7. Na janela Inspeção, clique no sinal de adição para expandir o gráfico.
8. Percorra as propriedades. Você está procurando propriedades que poderiam ter valores de 200, 10 e 15. Há uma propriedade `Elevation` com valor 10. Há uma propriedade `Rotation` com valor 200. Há uma configuração Perspectiva com valor 30. Essa configuração está maluca, porque o nome está exatamente certo, mas o valor é incorreto.
9. Volte para volta para o Excel. Escolha Rotação 3-D e mude a configuração Perspectiva de 15 para 20 na caixa de diálogo Formato.
10. Volte para o VBA e execute novamente a macro. Examine a janela Inspeção. A configuração Perspectiva mudou para 40. Você pode especular que essa é a propriedade certa, mas que o valor Perspectiva mostrado na caixa de diálogo Formato precisa ser duplicado quando inserido no VBA.

Você tem agora informações suficientes para escrever a seguinte macro para alterar a rotação do gráfico utilizando o VBA:

```
Sub RotateChart()
    Dim cht As Chart
    Set cht = ActiveChart
    cht.Rotation = 100
    cht.Elevation = 30
    cht.Perspective = 60 'Na verdade, significa 30%

End Sub
```

Criando gráficos avançados

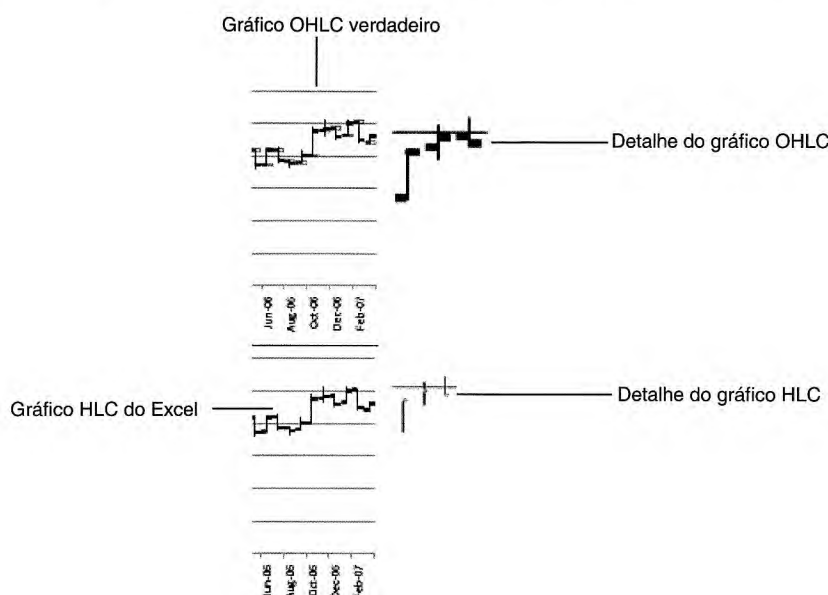
Em *Charts & Graphs for Microsoft Excel 2007*, mostrei alguns gráficos surpreendentes que parecem impossíveis de ser criados com o Excel. Criar esses gráficos normalmente envolve inserir uma série de dados não autorizados que aparece no gráfico como uma série de XY para completar algum efeito. O processo de criação desses gráficos manualmente é muito tedioso e garante que a maioria das pessoas nunca irá recorrer a ele.

Mas se você pudesse automatizar o processo de criação de gráficos utilizando o VBA, isso começaria a tornar-se viável.

NOVO Criando verdadeiros gráficos financeiros OHLC

Se você for fã de gráficos financeiros no *Wall Street Journal* ou finance.yahoo.com., reconhecerá o tipo de gráfico conhecido como OHLC (Open-High-Low-Close). O Microsoft Excel não oferece esse gráfico. No seu gráfico HLC (High-Low-Close) falta o traço voltado para a esquerda que representa a abertura de cada ponto. Agora, talvez você ache que os gráficos HLC são suficientemente parecidos com os gráficos OHLC, mas uma de minhas freqüentes reclamações é que o *WSJ* pode criar gráficos com uma aparência melhor do que o Excel.

Na Figura 11.12, você pode ver um gráfico OHLC verdadeiro acima da figura e um gráfico HLC do Excel na parte inferior.

Figura 11.12
O gráfico HLC predefinido do Excel omite a parte Open (de OHLC) em cada ponto de dados.

No Excel 2007, você pode especificar uma imagem personalizada a ser utilizada como o marcador em um gráfico. Fui imediatamente para o Photoshop e criei um traço voltado para a esquerda como um arquivo GIF. Esse pequeno elemento gráfico compensa o defeito fundamental na seleção de marcador de gráfico do Excel: o Excel oferece um traço voltado para a direita, mas não um para a esquerda. Você pode fazer o download do LeftDash.gif em www.mrexcel.com/getcode2007.html.

Na interface com o usuário Excel, você iria indicar que a série Open deve ter uma imagem personalizada e então especificar LeftDash.gif como a imagem. No código do VBA, você utiliza o método `UserPicture`, como mostrado aqui:

```
ActiveChart Cht.SeriesCollection(1).Fill.UserPicture ("C:\leftdash.gif")
```

Para criar um verdadeiro gráfico OHLC, siga estes passos:

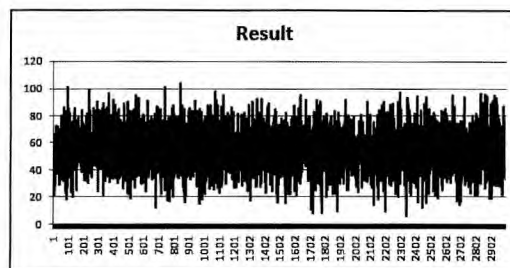
1. Crie um gráfico de linha de quatro séries; Open, High, Low, Close.
2. Mude o estilo da linha para Nenhum em todas as quatro séries.
3. Elimine o marcador da série High e Low.
4. Adicione uma linha High-Low ao gráfico.
5. Mude o marcador de Close para um traço voltado para a direita (chamado ponto em VBA) com tamanho 9.
6. Mude o marcador de Open para uma imagem personalizada e carregue LeftDash.gif como o preenchimento da série.

O código a seguir cria o gráfico superior na Figura 11.12:

```
Sub CreateOHCLChart()
    ' Baixe leftdash.gif do site do Mrexcel
    ' e salve-o na mesma pasta que essa pasta de trabalho
    Dim Cht As Chart
    Dim Ser As Series

    ActiveSheet.Shapes.AddChart(xlLineMarkers).Select
    Set Cht = ActiveChart
    Cht.SetSourceData Source:=Range("Sheet1!$A$1:$E$33")
    ' Formata a série Open
    With Cht.SeriesCollection(1)
        .MarkerStyle = xlMarkerStylePicture
        .Fill.UserPicture ("C:\leftdash.gif")
        .Border.LineStyle = xlNone
        .MarkerForegroundColorIndex = xlColorIndexNone
    End With
    ' Formata a série High & Low
    With Cht.SeriesCollection(2)
        .MarkerStyle = xlMarkerStyleNone
        .Border.LineStyle = xlNone
    End With
    With Cht.SeriesCollection(3)
        .MarkerStyle = xlMarkerStyleNone
        .Border.LineStyle = xlNone
    End With
    ' Formata a série Close
    Set Ser = Cht.SeriesCollection(4)
    With Ser
        .MarkerBackgroundColorIndex = 1
        .MarkerForegroundColorIndex = 1
        .MarkerStyle = xlDot
        .MarkerSize = 9
        .Border.LineStyle = xlNone
    End With
    ' Soma as linhas High-Low
    Cht.SetElement (msoElementLineHiLoLine)
    Cht.SetElement (msoElementLegendNone)

End Sub
```

Criando blocos para um gráfico de freqüência

Suponha que você tenha resultados de 3 mil avaliações científicas. Há uma boa maneira de produzir um gráfico desses resultados. Entretanto, se você só selecionar os resultados e criar um gráfico, acabará com um caos (veja Figura 11.13).

Figura 11.13
Tente organizar em gráfico o resultado das 3 mil avaliações e você terá um confuso conjunto de dados.

O truque para criar uma distribuição de freqüência efetiva é definir uma série de categorias ou *blocos*. Um função de array `FREQÜÊNCIA` conta o número de itens dos 3 mil resultados incluídos dentro de cada bloco.

O processo de criação de blocos manualmente é bastante tedioso e requer conhecimento de fórmulas de array. É melhor usar uma macro para fazer todos os cálculos tediosos.

A macro nesta seção requer a especificação de um tamanho de bloco (*bin*) e um bloco inicial. Se você espera resultados no intervalo de 0 a 100, poderia especificar blocos de 10 cada, iniciando em 0. Isso criaria blocos de 0–10, 11–20, 21–30 e assim por diante. Se especificar tamanhos de bloco de 15 com um bloco inicial de 5, a macro criará blocos de 5–20, 21–35, 36–50 e assim por diante.

Para utilizar o macro a seguir, seus resultados de avaliação devem iniciar na linha 2 e estar na coluna à direita de um conjunto de dados. Três variáveis próximas do começo da macro definem o bloco inicial, o bloco final e o tamanho de bloco:

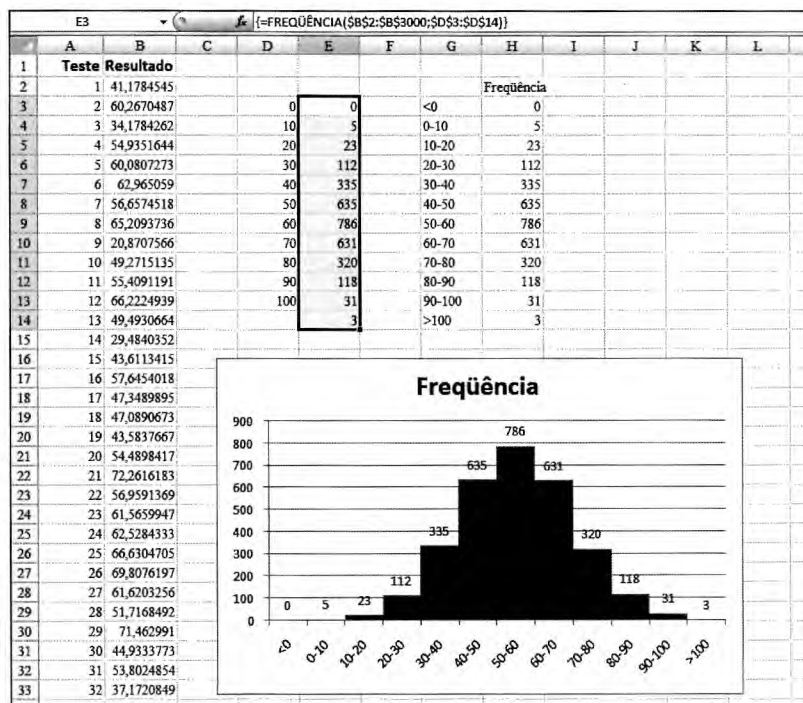
```
'Define os blocos
BinSize = 10
```

```
FirstBin = 0
LastBin = 100
```

Depois disso, a macro pula uma coluna e então cria um intervalo de blocos iniciais. Na célula D4 na Figura 11.14, o 10 é utilizado para dizer ao Excel que estão procurando o número de valores maior que 0 em D3, mas igual a ou menor que 10 em D4.

Embora os blocos se estendam a partir de D3:D13, a função `FREQÜÊNCIA` inserida na coluna E precisa incluir uma célula extra, caso alguns resultados sejam são maiores do que o último bloco. Essa única fórmula retorna muitos resultados. As fórmulas que retornam mais de uma resposta são chamadas *fórmulas de array*. Na interface com o usuário do Excel, você especifica uma fórmula de array mantendo as teclas Ctrl+Shift pressionadas enquanto pressiona Enter para terminar a fórmula. No Excel VBA, é preciso utilizar a propriedade `FormulaArray`. As próximas linhas da macro definem a fórmula de array na coluna E:

```
'Insere a fórmula Freqûência (aqui em inglês)
Form = "=FREQUENCY(R2C" & FinalCol & ":R" & FinalRow & "C" & FinalCol & _
    ",R3C" & NextCol & ":R" & _
    LastRow & "C" & NextCol & ")"
Range(Cells(FirstRow, NextCol + 1), Cells(LastRow, NextCol + 1)). _
    FormulaArray = Form
```

Figura 11.14
A macro resume os resultados em blocos e fornece um gráfico significativo dos dados.

Não é evidente para o leitor se o bloco indicado na coluna D é o limite superior ou inferior. A macro constrói rótulos legíveis na coluna G e então copia os resultados de freqüência na coluna H.

Depois que a macro constrói um gráfico de coluna simples, a linha seguinte elimina a lacuna entre colunas, criando a visualização tradicional de histograma dos dados:

```
Cht.ChartGroups(1).GapWidth = 0
```

A macro para criar o gráfico da Figura 11.14:

```
Sub CreateFrequencyChart()
    ' Localiza a última coluna
    FinalCol = Cells(1, Columns.Count).End(xlToLeft).Column
    ' Localiza o FinalRow
    FinalRow = Cells(Rows.Count, FinalCol).End(xlUp).Row

    ' Define os blocos
    BinSize = 10
    FirstBin = 0
    LastBin = 100
```

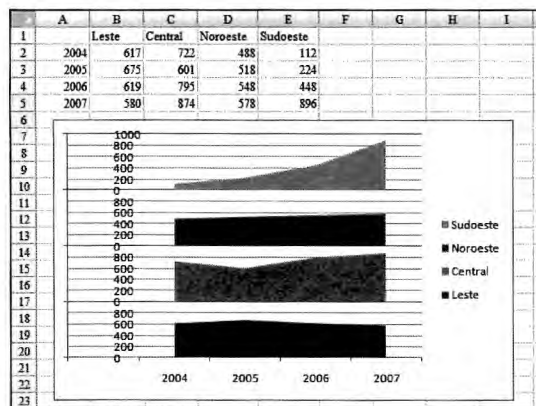
```
'Os blocos entrarão na linha 3, duas colunas depois da FinalCol
NextCol = FinalCol + 2
FirstRow = 3
NextRow = FirstRow - 1

' Configura os blocos para a função Freqüência
For i = FirstBin To LastBin Step BinSize
    NextRow = NextRow + 1
    Cells(NextRow, NextCol).Value = i
Next i

' A função Freqüência tem de ser uma linha maior que os blocos
LastRow = NextRow + 1

' Insere a fórmula Freqüência
Form = "=FREQUENCY(R2C" & FinalCol & ":R" & FinalRow & "C" & FinalCol & _
    ",R3C" & NextCol & ":R" & _
    LastRow & "C" & NextCol & ")"
Range(Cells(FirstRow, NextCol + 1), Cells(LastRow, NextCol + 1)). _
    FormulaArray = Form

' Cria um intervalo adequado para os dados de origem do gráfico
LabelCol = NextCol + 3
Form = "=R[-1]C[-3]&""-""&RC[-3]"
Range(Cells(4, LabelCol), Cells(LastRow - 1, LabelCol)).FormulaR1C1 = _
    Form
' Insere a última fórmula >
Cells(LastRow, LabelCol).FormulaR1C1 = "="">""&R[-1]C[-3]"
' Entre a primeira fórmula <
Cells(3, LabelCol).FormulaR1C1 = "=""<""&RC[-3]"

' Insere a fórmula para copiar os resultados de freqüência
Range(Cells(3, LabelCol + 1), Cells(LastRow, LabelCol + 1)).FormulaR1C1 = _
    "=RC[-3]"
' Adiciona um título
Cells(2, LabelCol + 1).Value = "Frequency"

' Cria um gráfico de colunas
Dim Cht As Chart
ActiveSheet.Shapes.AddChart(xlColumnClustered).Select
Set Cht = ActiveChart
Cht.SetSourceData Source:=Range(Cells(2, LabelCol), _
    Cells(LastRow, LabelCol + 1))
Cht.SetElement (msoElementLegendNone)
Cht.ChartGroups(1).GapWidth = 0
Cht.SetElement (msoElementDataLabelOutSideEnd)

End Sub
```

Criando um gráfico de área empilhada

É muito difícil criar o gráfico de área empilhada, mostrado na Figura 11.15, na interface com o usuário do Excel. Embora o gráfico pareça conter quatro gráficos independentes, ele na verdade contém nove séries:

- A primeira série contém os valores para a região Leste.
- A segunda série contém 1.000 menos os valores de Leste. Essa série é formatada com um preenchimento transparente.
- As séries 3, 5 e 7 contêm valores para as regiões Central, Noroeste e Sudoeste.
- As séries 4, 6 e 8 contêm 1.000 menos a série anterior.
- A série final é uma série XY, utilizada para adicionar rótulos ao eixo esquerdo. Há um ponto para cada linha de grade. Os marcadores são posicionados em uma posição X de 0. Rótulos de dados personalizados são adicionados ao lado de marcadores invisíveis para forçar os rótulos ao longo do eixo a iniciar novamente em 0 para cada região.

	Leste	Central	Noroeste	Sudoeste
2004	617	722	488	112
2005	675	601	518	224
2006	619	795	548	448
2007	580	874	578	896

Figura 11.15
Um único gráfico aparece para armazenar quatro gráficos diferentes.

Para utilizar a macro fornecida aqui, seus dados devem iniciar na coluna A e na linha 1. A macro adiciona novas colunas à direita dos dados e novas linhas abaixo deles; o resto da planilha deve estar em branco.

Duas variáveis na parte superior da macro definem a altura de cada gráfico. No exemplo atual, deixar uma altura de `1000` permite que as vendas de cada região se encaixem confortavelmente. O valor `LabSize` deve indicar a freqüência com que os rótulos devem aparecer ao longo do eixo esquerdo. Esse número deve ser igualmente divisível na altura do gráfico. Nesse exemplo, os valores `500`, `250`, `200`, `125` ou `100` funcionariam:

```
'Define a altura de cada gráfico de área
ChtHeight = 1000
'Define o tamanho da marca de seleção
'ChtHeight deve ser um múltiplo de LabSize
LabSize = 200
```

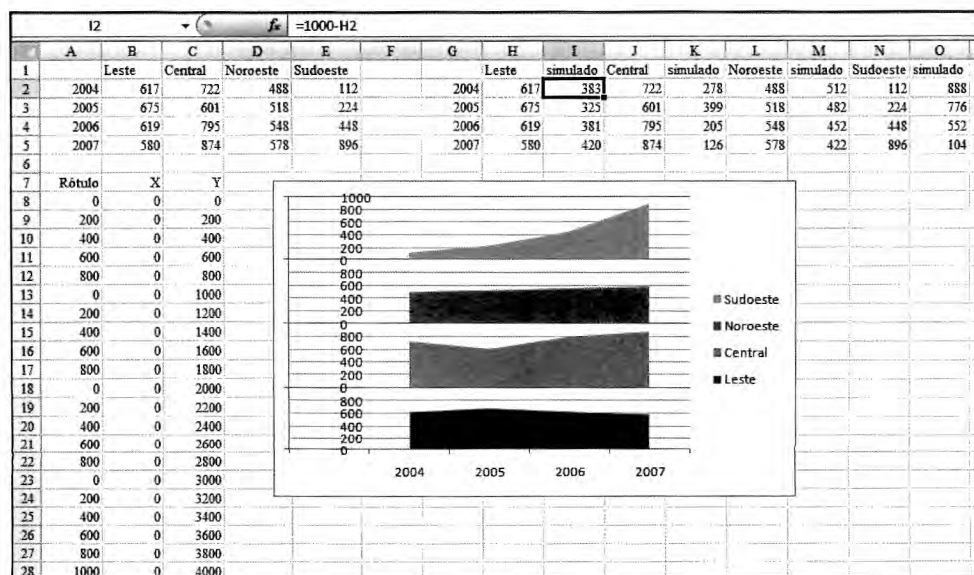
A macro cria uma cópia dos dados à direita dos dados originais. Novas séries 'simulado' são adicionadas à direita de cada região para calcular 1.000 menos o ponto de dados. Na Figura 11.16, essa série é mostrada em G1:O5.

A macro então cria um gráfico de área empilhada para as primeiras oito séries. A legenda para esse gráfico indica valores do Leste, simulado, Central, simulado e assim por diante. Para excluir uma entrada de legenda sim outra não, utilize esse código:

```
'Preenche a série 'simulado' sem preenchimento
For i = FinalSeriesCount To 2 Step -2
    Cht.SeriesCollection(i).Interior.ColorIndex = xlNone
Next i
```

I2 =1000-H2

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		Leste	Central	Noroeste	Sudoeste			Leste	simulado	Central	simulado	Noroeste	simulado	Sudoeste	simulado
2	2004	617	722	488	112		2004	617	383	722	278	488	512	112	888
3	2005	675	601	518	224		2005	675	325	601	399	518	482	224	776
4	2006	619	795	548	448		2006	619	381	795	205	548	452	448	552
5	2007	580	874	578	896		2007	580	420	874	126	578	422	896	104
6															
7	Rótulo	X	Y												
8	0	0	0												
9	200	0	200												
10	400	0	400												
11	600	0	600												
12	800	0	800												
13	0	0	1000												
14	200	0	1200												
15	400	0	1400												
16	600	0	1600												
17	800	0	1800												
18	0	0	2000												
19	200	0	2200												
20	400	0	2400												
21	600	0	2600												
22	800	0	2800												
23	0	0	3000												
24	200	0	3200												
25	400	0	3400												
26	600	0	3600												
27	800	0	3800												
28	1000	0	4000												

Figura 11.16
Os dados extras à direita e abaixo dos dados originais são criados pela macro a fim de criar o gráfico.

De maneira semelhante, o preenchimento para cada série de número no gráfico precisa ser configurada como transparente:

```
'Preenche a série 'simulado' sem preenchimento
For i = FinalSeriesCount To 2 Step -2
    Cht.SeriesCollection(i).Interior.ColorIndex = xlNone
Next i
```

A parte mais difícil do processo é adicionar uma nova série final ao gráfico. Essa série terá muito mais pontos de dados do que a outra série. O intervalo B8:C28 contém valores X e Y para nova série. Você verá que cada ponto tem um X valendo 0 para assegurar que ele aparecerá no lado esquerdo da área de plotagem. Os valores Y aumentam constantemente pelo valor indicado na variável `LabSize`. Na coluna A, ao lado dos pontos X e Y, estão os rótulos reais que serão plotados ao lado de cada marcador. Esses rótulos dão a ilusão de que o gráfico inicia com um valor 0 para cada região.

O processo de adicionar a nova série é realmente muito mais fácil no VBA do que na interface com o usuário do Excel. O código a seguir identifica cada componente da série e especifica que deve ser plotado como um gráfico XY:

```
'Adiciona a nova série ao gráfico
Set Ser = Cht.SeriesCollection.NewSeries
With Ser
    .Name = "Y"
    .Values = Range(Cells(AxisRow + 1, 3), Cells(NewFinal, 3))
    .XValues = Range(Cells(AxisRow + 1, 2), Cells(NewFinal, 2))
    .ChartType = xlXYScatter
    .MarkerStyle = xlMarkerStyleNone
End With
```

Por fim, o código do início deste capítulo aplica um rótulo de dados da coluna A a cada ponto na série final:

```
'Rotula cada ponto na série
'Esse código realmente adiciona rótulos falsos no eixo esquerdo
For i = 1 To TickMarkCount
    Ser.Points(i).HasDataLabel = True
    Ser.Points(i).DataLabel.Text = Cells(AxisRow + i, 1).Value
Next i
```

O código completo para criar o gráfico empilhado na Figura 11.16 é mostrado aqui:

```
Sub CreatedStackedChart()
    Dim Cht As Chart
    Dim Ser As Series
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    FinalCol = Cells(1, Columns.Count).End(xlToLeft).Column
    OrigSeriesCount = FinalCol - 1
    FinalSeriesCount = OrigSeriesCount * 2

    ' Define a altura de cada gráfico de área
    ChtHeight = 1000
    ' Define o tamanho da marca de seleção
    ' ChtHeight deve ser múltiplo de LabSize
    LabSize = 200

    ' Faz uma cópia dos dados
    NextCol = FinalCol + 2
    Cells(1, 1).Resize(FinalRow, FinalCol).Copy _
        Destination:=Cells(1, NextCol)
    FinalCol = Cells(1, Columns.Count).End(xlToLeft).Column

    ' Adiciona novas colunas para servir como série 'simulado'
    MyFormula = "=" & ChtHeight & "-RC[-1]"
    For i = FinalCol + 1 To NextCol + 2 Step -1
        Cells(1, i).EntireColumn.Insert
        Cells(1, i).Value = "simulado"
        Cells(2, i).Resize(FinalRow - 1, 1).FormulaR1C1 = MyFormula
    Next i

    ' Descobre a nova coluna final
    FinalCol = Cells(1, Columns.Count).End(xlToLeft).Column

    ' Cria o gráfico
    ActiveSheet.Shapes.AddChart(xlAreaStacked).Select
    Set Cht = ActiveChart
    Cht.SetSourceData Source:=Range(Cells(1, NextCol), Cells(FinalRow, FinalCol))
    Cht.PlotBy = xlColumns

    ' Limpa a série de números pares da Legenda
    For i = FinalSeriesCount - 1 To 1 Step -2
        Cht.Legend.LegendEntries(i).Delete
    Next i
```

```
    ' Configura o eixo Escala Máxima & Linhas de Grade
    TopScale = OrigSeriesCount * ChtHeight
    With Cht.Axes(xlValue)
        .MaximumScale = TopScale
        .MinorUnit = LabSize
        .MajorUnit = ChtHeight
    End With
    Cht.SetElement (msoElementPrimaryValueGridLinesMinorMajor)

    ' Preenche a série 'simulado' sem preenchimento
    For i = FinalSeriesCount To 2 Step -2
        Cht.SeriesCollection(i).Interior.ColorIndex = xlNone
    Next i

    ' Oculta os rótulos do eixo original
    Cht.Axes(xlValue).TickLabelPosition = xlNone

    ' Cria um novo intervalo para armazenar uma série XY não autorizada que será
    ' utilizada para criar rótulos no eixo esquerdo
    AxisRow = FinalRow + 2
    Cells(AxisRow, 1).Resize(1, 3).Value = Array("Label", "X", "Y")
    TickMarkCount = OrigSeriesCount * (ChtHeight / LabSize) + 1
    ' A coluna B contém os valores X. Esses são todos zeros
    Cells(AxisRow + 1, 2).Resize(TickMarkCount, 1).Value = 0
    ' A coluna C contém os valores Y.
    Cells(AxisRow + 1, 3).Resize(TickMarkCount, 1).FormulaR1C1 = _
        "=R[-1]C+" & LabSize
    Cells(AxisRow + 1, 3).Value = 0
    ' A coluna A contém os rótulos a ser usados para cada ponto
    Cells(AxisRow + 1, 1).Value = 0
    Cells(AxisRow + 2, 1).Resize(TickMarkCount - 1, 1).FormulaR1C1 = _
        "=IF(R[-1]C+" & LabSize & ">=" & ChtHeight & ",0,R[-1]C+" & LabSize & ")"
    NewFinal = Cells(Rows.Count, 1).End(xlUp).Row
    Cells(NewFinal, 1).Value = ChtHeight

    ' Adiciona a nova série ao gráfico
    Set Ser = Cht.SeriesCollection.NewSeries
    With Ser
        .Name = "Y"
        .Values = Range(Cells(AxisRow + 1, 3), Cells(NewFinal, 3))
        .XValues = Range(Cells(AxisRow + 1, 2), Cells(NewFinal, 2))
        .ChartType = xlXYScatter
        .MarkerStyle = xlMarkerStyleNone
    End With

    ' Rotula cada ponto na série
    ' Esse código realmente adiciona rótulos falsos no eixo esquerdo
    For i = 1 To TickMarkCount
        Ser.Points(i).HasDataLabel = True
        Ser.Points(i).DataLabel.Text = Cells(AxisRow + i, 1).Value
    Next i

    ' Oculta o rótulo Y na legenda
    Cht.Legend.LegendEntries(Cht.Legend.LegendEntries.Count).Delete
End Sub
```

Os sites Web de Andy Pope (www.andypope.info) e Jon Peltier (peltiertech.com/) apresentam exemplos de gráficos incomuns, que requerem um esforço extraordinário para serem criados. Se tiver de criar regularmente gráficos empilhados ou qualquer outro gráfico como os daqueles sites, reservar tempo para escrever o VBA aliviará do árduo trabalho de criar gráficos na interface com o usuário do Excel.

Exportando um gráfico como um elemento gráfico

Você pode exportar qualquer gráfico para um arquivo de imagem na unidade de disco. O método `ExportChart` requer que se especifique um nome de arquivo e um tipo gráfico. Os tipos gráficos disponíveis dependem de filtros de arquivos gráficos instalados em seu Registro. É uma aposta segura que JPG, BMP, PNG e GIF funcionarão na maioria dos computadores.

Por exemplo, o código a seguir exporta o gráfico ativo como um arquivo GIF:

```
Sub ExportChart()
    Dim cht As Chart
    Set cht = ActiveChart
    cht.Export Filename:="C:\Chart.gif", Filtername:="GIF"
End Sub
```

NOTA

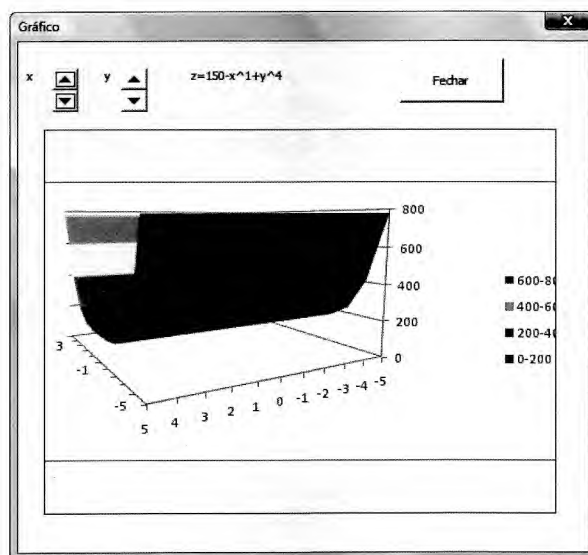
Desde o Excel 2003, a Microsoft suporta um argumento Interativo no método Export. A ajuda do Excel indica que, se você configurou Interativo como VERDADEIRO, o Excel pedirá configurações adicionais dependendo do tipo de arquivo. Entretanto, a caixa de diálogo para solicitar configurações adicionais nunca aparecerá, pelo menos não para os quatro tipos-padrão de JPG, GIF, BMP ou PNG.

Criando um gráfico dinâmico em um userform

Com a capacidade de exportar um gráfico para um arquivo gráfico, também há a capacidade de carregar um arquivo gráfico em um controle `Image` de um userform. Isso significa que podemos criar uma caixa de diálogo em que alguém pode controlar dinamicamente os valores utilizados para a plotagem de um gráfico.

Para criar a caixa de diálogo mostrada na Figura 11.17, siga estes passos:

1. Na janela VBA, escolha Inserir, UserForm. Na janela Propriedades, renomeie o formulário **`frmChart`**.
2. Redimensione o userform.
3. Adicione um grande controle `Image` ao userform.
4. Adicione dois botões seletores chamados **`sbX`** e **`sbY`**. Configure-os para ter no mínimo `1` e no máximo `5`.

Figura 11.17
Essa caixa de diálogo é um userform VBA exibindo um gráfico. O gráfico redefine-se com base em alterações nos controles da caixa de diálogo.

5. Adicione um controle `Label3` para exibir a fórmula.
6. Adicione um botão de comando com o rótulo `Fechar`.
7. Insira esse código na janela Código atrás do formulário:

```
Private Sub CommandButton1_Click()
    Unload Me
End Sub
Private Sub sbX_Change()
    Worksheets("Superficie").Range("02").Value = Me.sbX.Value
    Worksheets("Superficie").Shapes(1).Chart.Export "C:\Chart.gif"
    Me.Label3.Caption = Worksheets("Superficie").Range("04").Value
    Me.Image1.Picture = LoadPicture("C:\Chart.gif")
End Sub
Private Sub sbY_Change()
    Worksheets("Superficie").Range("03").Value = Me.sbY.Value
    Worksheets("Superficie").Shapes(1).Chart.Export "C:\Chart.gif"
```

```
    Me.Label3.Caption = Worksheets("Superficie").Range("O4").Value
    Me.Image1.Picture = LoadPicture("C:\Chart.gif")
End Sub

Private Sub UserForm_Initialize()
    Me.sbX = Worksheets("Superficie").Range("O2").Value
    Me.sbY = Worksheets("Superficie").Range("O3").Value
    Me.Label3.Caption = Worksheets("Superficie").Range("O4").Value
    Worksheets("Superficie").Shapes(1).Chart.Export "C:\Chart.gif"
    Me.Image1.Picture = LoadPicture("C:\Chart.gif")
End Sub
```

8. Utilize Inserir, Módulo para adicionar um componente de Module1 com este código:

```
Sub ShowForm()
    frmChart.Show
End Sub
```

À medida que alguém altera os botões seletores no userform, o Excel escreve novos valores na planilha. Isso atualiza o gráfico. O código de userform então exporta o gráfico e o exibe no userform (consulte a Figura 11.17).

Criando gráficos dinâmicos

Um gráfico dinâmico é aquele que utiliza uma tabela dinâmica como a origem de dados subjacente. Infelizmente, os gráficos dinâmicos não têm a excelente funcionalidade de 'mostrar páginas' presente nas tabelas dinâmicas. Você pode superar esse problema com uma macro VBA rápida que cria uma tabela dinâmica e depois um gráfico dinâmico baseado na tabela dinâmica. A macro então adiciona o campo de cliente à área de filtro de relatório da tabela dinâmica. Em seguida, faz loop por cada cliente e exporta o gráfico para cada cliente.

No Excel 2007, primeiro você cria um cache dinâmico utilizando o método `PivotCache.Create`. Depois pode definir uma tabela dinâmica baseada no cache dinâmico. O procedimento comum é desativar a atualização da tabela dinâmica enquanto estiver adicionando campos à tabela dinâmica. Então você atualiza essa tabela para que o Excel realize os cálculos.

É preciso um pouco de astúcia para descobrir o intervalo final da tabela dinâmica. Se você desativar os totais da coluna e da linha, a área de gráfico da tabela dinâmica iniciará uma linha abaixo da área `PivotTableRange1`. Você terá de redimensionar a área para incluir uma linha a menos para que seu gráfico apareça corretamente.

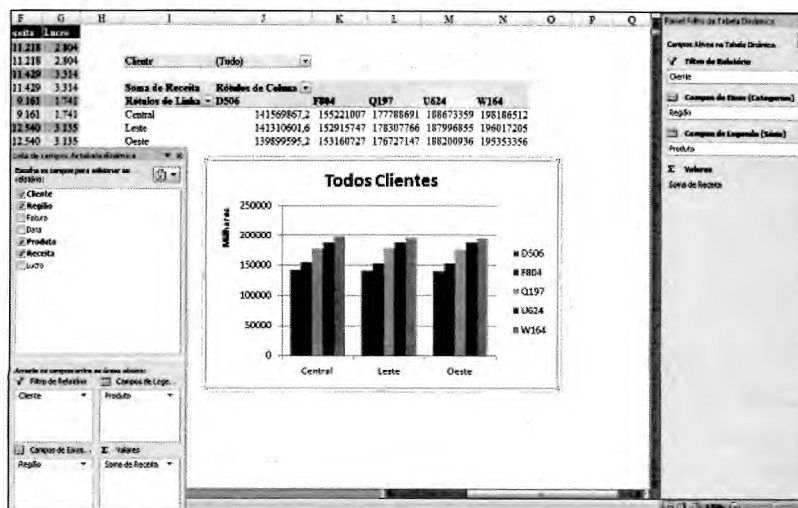
Depois que a tabela dinâmica é criada, você pode voltar para o código `Charts.Add` discutido anteriormente neste capítulo. Pode utilizar qualquer código de formatação para obter o gráfico formatado como você quiser.

O código a seguir cria uma tabela dinâmica e um único gráfico dinâmico que resume a renda por região e produto:

```
Sub CreateSummaryReportUsingPivot()
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long
    Dim ChartDataRange As Range
    Dim Cht As Chart
    Set WSD = Worksheets("Dados")

    ' Exclui todas as tabelas dinâmicas anteriores
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("I1:Z1").EntireColumn.Clear

    ' Define a área de entrada e configura um cache dinâmico
    FinalRow = WSD.Cells(Application.Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Application.Columns.Count). _
        End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)

    Set PTCache = ActiveWorkbook.PivotCaches.Create(SourceType:= _
        xlDatabase, SourceData:=PRange.Address)

    ' Cria a tabela dinâmica a partir do cache dinâmico
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD. _
        Cells(2, FinalCol + 2), TableName:="PivotTable1")
```

```
    ' Desativa a atualização ao criar a tabela
    PT.ManualUpdate = True

    ' Configura os campos de linha
    PT.AddFields RowFields:="Região", ColumnFields:="Produto", _
        PageFields:="Cliente"

    ' Configura os campos de dados
    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
    End With

    With PT
        .ColumnGrand = False
        .RowGrand = False
        .NullString = "0"
    End With

    ' Calcula a tabela dinâmica
    PT.ManualUpdate = False
    PT.ManualUpdate = True

    ' Define o intervalo de dados do gráfico
    Set ChartDataRange = _
        PT.TableRange1.Offset(1, 0).Resize(PT.TableRange1.Rows.Count - 1)

    ' Adiciona o gráfico
    WSD.Shapes.AddChart.Select
    Set Cht = ActiveChart
    Cht.SetSourceData Source:=ChartDataRange
    ' Formata o gráfico
    Cht.ChartType = xlColumnClustered
    Cht.SetElement (msoElementChartTitleAboveChart)
    Cht.ChartTitle.Caption = "Todos Clientes"
    Cht.SetElement msoElementPrimaryValueAxisThousands
End Sub
```

A Figura 11.18 mostra o gráfico e a tabela dinâmica resultantes.

Figura 11.18
O VBA cria uma tabela dinâmica e então um gráfico a partir dessa tabela. O Excel exibe automaticamente a janela Filtro da Tabela Dinâmica em resposta.

Próximos passos

Os gráficos fornecem uma imagem visual que pode ajudar a resumir dados para um gerente. No Capítulo 12, “Exploração de dados com o filtro avançado”, você aprenderá como utilizar as ferramentas de filtro avançado para produzir relatórios rapidamente.

Exploração de dados com o filtro avançado

12

NESTE CAPÍTULO

Filtro Avançado é mais fácil em VBA que no Excel

É tão difícil usar o misterioso comando Filtro Avançado na interface com o usuário do Excel que raras vezes encontramos alguém que goste de usá-lo regularmente. Nas versões do Excel anteriores à versão 2007, é provável que o comando AutoFiltro fosse muito mais utilizado que o Filtro Avançado. No Excel 2007, a Microsoft renomeou o AutoFiltro como Filtro e fez melhorias nos possíveis tipos de filtro, tornando o Filtro Avançado uma escolha menos possível na interface com o usuário.

Mas, no VBA, é muito fácil utilizar os filtros avançados. Com apenas uma linha de código, podemos extrair rapidamente um subconjunto de registros de um banco de dados ou obter uma lista de valores únicos em qualquer coluna. Isso é importante quando queremos fazer relatórios para uma determinada região ou cliente.

Como poucas pessoas utilizam o recurso Filtro Avançado, dou exemplos passo a passo, utilizando a interface com o usuário para criar um filtro avançado e mostrar código análogo. Você ficará surpreso em ver como a interface com o usuário parece complexa e que, apesar disso, é fácil programar um poderoso filtro avançado para extrair registros.

Uma razão pela qual o Filtro Avançado é difícil de utilizar é que existem várias maneiras diferentes de usá-lo. Você deve fazer três escolhas básicas na caixa de diálogo Filtro Avançado. Como cada escolha tem duas opções, há oito (2 x 2 x 2) possíveis combinações dessas escolhas. As três escolhas são mostradas na Figura 12.1 e descritas aqui:

- **Ação** — Você pode escolher Filtrar a Lista no Local ou Copiar Para Outro Local. Se escolher Filtrar os Registros no Local, as linhas não-correspondentes serão ocultadas. Escolher Copiar Para Outro Local copia os registros que correspondem ao filtro para um novo intervalo.
- **Critérios** — Você pode filtrar com ou sem critérios. Filtrar com critérios é apropriado para obter um subconjunto de linhas. Filtrar sem critério é útil quando você quer um subconjunto de colunas ou está utilizando a opção Somente Registros Exclusivos.
- **Único** — Você pode escolher se solicita Somente Registros Exclusivos ou todos os registros correspondentes. A primeira opção torna o comando Filtro Avançado uma das formas mais rápidas para localizar uma lista de valores únicos em um campo.

Figura 12.1
É complicado usar a caixa de diálogo Filtro Avançado na interface com o usuário do Excel. Felizmente, isso é muito fácil no VBA.

Utilizando Filtro Avançado para extrair uma lista de valores únicos

Um dos usos mais simples do Filtro Avançado é extrair uma lista única de um único campo de um conjunto de dados. Neste exemplo, queremos obter uma lista única de clientes de um relatório de vendas. Você sabe que esse cliente está na Coluna D do conjunto de dados. Há um número desconhecido de registros que iniciam na célula A2. (A linha 1 é a linha de cabeçalho.) Não há nada à direita do conjunto de dados.

Extraindo uma lista de valores únicos com a interface com o usuário

Para extrair uma lista de valores únicos, siga estes passos:

1. Com o cursor em qualquer lugar do intervalo de dados, selecione Avançado do grupo Classificar & Filtrar na faixa Dados. Na primeira vez que utilizar o comando Filtro Avançado em uma planilha, o Excel preencherá automaticamente a caixa de texto Intervalo da Lista com o intervalo inteiro do seu conjunto de dados. Nas utilizações subseqüentes do comando Filtro Avançado, essa caixa de diálogo nos lembra das configurações de filtro avançado anteriores.
2. Escolha a caixa de seleção Somente Registros Exclusivos na parte inferior da caixa de diálogo.
3. Na seção Ação, escolha Copiar Para Outro Local.
4. Digite `J1` na caixa de texto Copiar Para.

Por padrão, o Excel copia todas as colunas no conjunto de dados. Você pode filtrar apenas a coluna Cliente, limitando o Intervalo da Lista para incluir apenas a Coluna D ou especificando um ou mais títulos no intervalo Copiar Para. Qualquer um desses métodos tem suas próprias desvantagens.

Mude o Intervalo da Lista para uma Única Coluna

Edite o Intervalo da Lista para apontar para a coluna Cliente. Nesse caso, isso significa mudar o padrão `$A$1:$H$1127` para `$D$1:$D$1127`. A caixa de diálogo Filtro Avançado deve aparecer.

DICA

Quando você edita inicialmente qualquer intervalo na caixa de diálogo, o Excel pode estar no modo Ponto. Nesse modo, pressionar uma tecla de seta para esquerda ou para a direita inserirá uma referência de célula na caixa de texto. Se você vir a palavra 'Ponto' no canto inferior esquerdo da janela do Excel, pressione a tecla F2 para mudar o modo Ponto para o modo de Edição.

A desvantagem desse método é que o Excel se lembra do intervalo da lista nas utilizações subseqüentes do comando Filtro Avançado. Se, mais tarde, você quiser obter uma lista exclusiva de regiões, terá de constantemente especificar o intervalo da lista.

Mude o Intervalo da Lista para uma Única Coluna

Fazendo uma pequena previsão antes de invocar o comando Filtro Avançado, você pode permitir que o Excel mantenha o intervalo de lista padrão de `$A$1:$H$1127`. Na célula J1, digite o título `Cliente`. Na Figura 12.2, você deixa o campo Intervalo da Lista apontando para as colunas de A a H. Como o intervalo Copiar Para de J1 já contém um título válido no intervalo da lista, o Excel copia os dados exclusivamente da coluna Cliente. Prefiro esse método, particularmente ao fazer múltiplos filtros avançados. Como o Excel se lembra das configurações anteriores do último filtro avançado, é mais conveniente filtrar sempre as colunas inteiras do Intervalo da Lista e limitar as colunas, configurando títulos no intervalo Copiar Para.

Figura 12.2
Configurando um título no intervalo Copiar Para de J1, você pode evitar ter de especificar novamente o intervalo da lista para cada filtro avançado.

Depois de utilizar qualquer um desses métodos para fazer o Filtro Avançado, uma lista concisa dos clientes únicos aparece na coluna J (veja Figura 12.3).

Figura 12.3
O Filtro Avançado extraiu uma lista única de clientes do conjunto de dados e a copiou na coluna J.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Região	Produto	Data	Cliente	Quantidade	Receita	Custo Vendas	Lucro		Cliente		
2	Leste	R537	24-Jul-07	Trustworthy Flagpole Partners	1000	22810	11242	11568		Trustworthy Flagpole Partners		
3	Leste	M556	25-Jul-07	Amazing Shoe Company	500	10245	4659	5586		Amazing Shoe Company		
4	Central	W435	25-Jul-07	Amazing Shoe Company	100	2257	1082	1175		Mouthwatering Notebook Inc.		
5	Central	R537	26-Jul-07	Mouthwatering Notebook Inc.	500	11240	5621	5619		Cool Saddle Traders		
6	Leste	R537	27-Jul-07	Cool Saddle Traders	400	9152	4497	4655		Tasty Shovel Company		
7	Leste	W435	27-Jul-07	Tasty Shovel Company	800	18552	8659	9893		Distinctive Wax Company		
8	Central	R537	27-Jul-07	Mouthwatering Notebook Inc.	400	9204	4497	4707		Guarded Aerobic Corporation		
9	Central	M556	28-Jul-07	Distinctive Wax Company	400	6860	3727	3133		Tasty Yogurt Corporation		
10	Leste	M556	30-Jul-07	Guarded Aerobic Corporation	400	8456	3727	4729		Agile Aquarium Inc.		
11	Oeste	R537	30-Jul-07	Cool Saddle Traders	600	13806	6745	7061		Magnificent Eggbeater Corporation		
12	Leste	W435	30-Jul-07	Tasty Yogurt Corporation	1000	21730	10824	10906		User-Friendly Luggage Corporation		
13	Central	M556	1-Aug-07	Guarded Aerobic Corporation	800	16416	7454	8962		Guarded Umbrella Traders		
14	Leste	R537	1-Aug-07	Agile Aquarium Inc.	900	21015	10118	10897		Enhanced Eggbeater Corporation		
15	Central	R537	2-Aug-07	Magnificent Eggbeater Corporation	900	21438	10118	11320		Honest Shoe Supply		
16	Leste	R537	2-Aug-07	Mouthwatering Notebook Inc.	900	21465	10118	11347		First-Rate Notebook Inc.		

Extrair uma lista de valores únicos com o código VBA

No VBA, você utiliza o método `AdvancedFilter` para executar o comando Filtro Avançado. Novamente, há três escolhas:

- **Ação** — Escolha filtrar no local com o parâmetro `Action:=xlFilterInPlace` ou copiar com `Action:=xlFilterCopy`. Se quiser copiar, você também tem de especificar o parâmetro `CopyToRange:=Range("J1")`.
- **Critérios** — Para filtrar com critérios, inclua o parâmetro `CriteriaRange:=Range("L1:L2")`. Para filtrar sem critérios, omita esse parâmetro opcional.
- **Único** — Para retornar exclusivamente os registros únicos, especifique o parâmetro `Unique:=True`.

O próximo código define um intervalo de saída de uma única saída duas colunas à direita da última coluna utilizada no intervalo de dados:

```
Sub GetUniqueCustomers()
    Dim IRange As Range
    Dim ORange As Range

    ' Localiza o tamanho do conjunto de dados de hoje
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    NextCol = Cells(1, Columns.Count).End(xlToLeft).Column + 2

    ' Configura intervalo de resultados. Copia o título de D1
    Range("D1").Copy Destination:=Cells(1, NextCol)
    Set ORange = Cells(1, NextCol)

    ' Define o intervalo de entrada
    Set IRange = Range("A1").Resize(FinalRow, NextCol - 2)

    ' Faz o filtro avançado obter a lista única de clientes
    IRange.AdvancedFilter Action:=xlFilterCopy, CopyToRange:=ORange, Unique:=True

End Sub
```

Por padrão, um filtro avançado copia todas as colunas. Se você só quer uma determinada coluna, utilize o título dessa coluna como título do intervalo de saída.

O primeiro fragmento de código localiza a linha final e a coluna no conjunto de dados. Embora não seja necessário fazer isso, defino uma variável de objeto para o intervalo de saída (`ORange`) e para o intervalo de entrada (`IRange`).

Esse código é genérico o bastante para não precisar ser regravado se novas colunas forem adicionadas posteriormente ao conjunto de dados. A configuração das variáveis de objeto para os intervalos de entrada e de saída é feita por uma questão de legibilidade e não de necessidade.

O código anterior poderia ser escrito com a mesma facilidade que esta versão abreviada:

```
Sub UniqueCustomerRedux()
    ' Copia um título para criar um intervalo de saída
    Range("J1").Value = Range("D1").Value
    ' Aplica o filtro avançado
    Range("A1").CurrentRegion.AdvancedFilter xlFilterCopy, _
        CopyToRange:=Range("J1"), Unique:=True
End Sub
```

Ao executar qualquer um dos blocos de código anteriores no conjunto de dados de exemplo, você fecha uma lista única de clientes à direita dos dados. Na Figura 12.3, você viu o conjunto de dados original nas Colunas A:H e os clientes únicos na Coluna J. A chave para obter uma lista única de clientes é copiar o cabeçalho do campo Cliente para uma célula em branco e especificar essa célula como intervalo da saída.

Depois de obter a lista única de clientes, você pode facilmente classificá-la e adicionar uma fórmula `SUMIF` para obter a renda total por cliente. O próximo código obtém a lista única de clientes, classifica-a e cria uma fórmula para somar a renda por cliente. A Figura 12.4 mostra os resultados:

```
Sub RevenueByCustomers()
    Dim IRange As Range
    Dim ORange As Range

    ' Localiza o tamanho do conjunto de dados de hoje
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    NextCol = Cells(1, Columns.Count).End(xlToLeft).Column + 2

    ' Configura intervalo de resultados. Copia o título de D1
    Range("D1").Copy Destination:=Cells(1, NextCol)
    Set ORange = Cells(1, NextCol)

    ' Define o intervalo de entrada
    Set IRange = Range("A1").Resize(FinalRow, NextCol - 2)

    ' Faz o filtro avançado obter a lista única de clientes
    IRange.AdvancedFilter Action:=xlFilterCopy, _
        CopyToRange:=ORange, Unique:=True

    ' Determina quantos clientes únicos temos
    LastRow = Cells(Rows.Count, NextCol).End(xlUp).Row

    ' Classifica os dados
    Cells(1, NextCol).Resize(LastRow, 1).Sort Key1:=Cells(1, NextCol), _
        Order1:=xlAscending, Header:=xlYes

    ' Adiciona uma fórmula SOMASE (aqui em inglês) para obter totais
    Cells(1, NextCol + 1).Value = "Receita"
    Cells(2, NextCol + 1).FormulaR1C1 = "=SUM(R2C4:R" & FinalRow & "C4,RC[-1], _
        R2C6:R" & FinalRow & "C6)"
    If LastRow > 2 Then
        Cells(2, NextCol + 1).Copy Cells(3, NextCol + 1).Resize(LastRow - 2, 1)
    End If

End Sub
```

{=SOMA(($D$2:$D$1127=J2)*$F$2:$F$1127)}

J	K
Cliente	Receita
Agile Aquarium Inc.	97107
Amazing Shoe Company	820384
Appealing Eggbeater Corporation	92544
Cool Saddle Traders	53170
Distinctive Wax Company	947025
Enhanced Eggbeater Corporation	1543677
First-Rate Glass Corporation	106442
First-Rate Notebook Inc.	104205
Guarded Aerobic Corporation	1448081

Figura 12.4
Essa macro simples produziu um relatório sumário por cliente a partir de um extenso conjunto de dados. Utilizar `AdvancedFilter` é a chave para macros poderosas como essas.

Outra utilização para uma lista de valores únicos é o preenchimento rápido de uma caixa de listagem ou caixa de combinação em um userform. Suponha, por exemplo, que você tem uma macro que pode executar um relatório para um cliente específico qualquer. Para permitir que seus usuários escolham para quais clientes serão feitos relatórios, crie um userform simples. Adicione uma caixa de listagem ao userform e configure a propriedade `MultiSelect` da caixa de listagem como `1-fmMultiSelectMulti`. Nomeei meu formulário como frmReport. Além da caixa de listagem, tenho quatro botões de comando: OK, Cancelar, Marcar Tudo, Limpar Tudo. A seguir, apresento o código para executar o formulário. Note que o procedimento `Userform_Initialize` inclui um filtro avançado para obter a lista única de clientes do conjunto de dados:

```
Private Sub CancelButton_Click()
    Unload Me
End Sub

Private Sub cbSubAll_Click()
    For i = 0 To lbCust.ListCount - 1
        Me.lbCust.Selected(i) = True
    Next i
End Sub
```

```
Private Sub cbSubClear_Click()
    For i = 0 To lbCust.ListCount - 1
        Me.lbCust.Selected(i) = False
    Next i
End Sub

Private Sub OKButton_Click()
    For i = 0 To lbCust.ListCount - 1
        If Me.lbCust.Selected(i) = True Then
            ' Chama uma rotina para produzir esse relatório
            RunCustReport WhichCust:=Me.lbCust.List(i)
        End If
    Next i
    Unload Me
End Sub

Private Sub UserForm_Initialize()
    Dim IRange As Range
    Dim ORange As Range

    ' Localiza o tamanho do conjunto de dados de hoje
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    NextCol = Cells(1, Columns.Count).End(xlToLeft).Column + 2

    ' Configura intervalo de resultados. Copia o título de D1
    Range("D1").Copy Destination:=Cells(1, NextCol)
    Set ORange = Cells(1, NextCol)

    ' Define o intervalo de entrada
    Set IRange = Range("A1").Resize(FinalRow, NextCol - 2)

    ' Faz o filtro avançado obter a lista única de clientes
    IRange.AdvancedFilter Action:=xlFilterCopy, _
        CopyToRange:=ORange, Unique:=True

    ' Determina quantos clientes únicos temos
    LastRow = Cells(Rows.Count, NextCol).End(xlUp).Row

    ' Classifica os dados
    Cells(1, NextCol).Resize(LastRow, 1).Sort Key1:=Cells(1, NextCol), _
        Order1:=xlAscending, Header:=xlYes

With Me.lbCust
    .RowSource = ""
    .List = Cells(2, NextCol).Resize(LastRow - 1, 1).Value
End With


    ' Apaga a lista temporária de clientes
    Cells(1, NextCol).Resize(LastRow, 1).Clear
End Sub
```

Carregue esse formulário com um módulo simples como este:

```
Sub ShowCustForm()
    frmReport.Show
End Sub
```

Seus clientes irão receber uma lista de todos os clientes válidos do conjunto de dados. Como a propriedade `MultiSelect` da caixa de listagem é configurada para permitir isso, eles poderão selecionar qualquer número de clientes, como mostrado na Figura 12.5.

Obtendo combinações únicas de dois ou mais campos

Para obter todas as combinações únicas de dois (ou mais) campos, crie o intervalo de saída para os campos adicionais. Essa amostra de código cria uma lista de combinações únicas de dois campos, Cliente e Produto:

```
Sub UniqueCustomerProduct()
    Dim IRange As Range
    Dim ORange As Range
```

```
    ' Localiza o tamanho do conjunto de dados de hoje
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    NextCol = Cells(1, Columns.Count).End(xlToLeft).Column + 2

    ' Configura o intervalo de resultados. Copia os títulos de D1 & B1
    Range("D1").Copy Destination:=Cells(1, NextCol)
    Range("B1").Copy Destination:=Cells(1, NextCol + 1)
    Set ORange = Cells(1, NextCol).Resize(1, 2)

    ' Define o intervalo de entrada
    Set IRange = Range("A1").Resize(FinalRow, NextCol - 2)

    ' Faz o filtro avançado obter a lista única de clientes & produto
    IRange.AdvancedFilter Action:=xlFilterCopy, _
        CopyToRange:=ORange, Unique:=True

    ' Determina quantas linhas únicas temos
    LastRow = Cells(Rows.Count, NextCol).End(xlUp).Row

    ' Classifica os dados
    Cells(1, NextCol).Resize(LastRow, 2).Sort Key1:=Cells(1, NextCol), _
        Order1:=xlAscending, Key2:=Cells(1, NextCol + 1), _
        Order2:=xlAscending, Header:=xlYes

End Sub
```

No resultado mostrado na Figura 12.6, você pode ver que a Enhanced Eggbeater compra apenas um produto e a Agile Aquarium compra três produtos.

Figura 12.5
Os clientes poderão fazer rápidas seleções nessa lista. Utilizar um filtro avançado em um conjunto de dados de 1 milhão de linhas é muito mais rápido do que configurar uma classe para preencher a caixa de listagem.

J	K
Cliente	Produto
Agile Aquarium Inc.	M556
Agile Aquarium Inc.	R537
Agile Aquarium Inc.	W435
Amazing Shoe Company	M556
Amazing Shoe Company	R537
Amazing Shoe Company	W435
Appealing Eggbeater Corporation	M556
Appealing Eggbeater Corporation	R537
Appealing Eggbeater Corporation	W435
Cool Saddle Traders	M556
Cool Saddle Traders	R537
Cool Saddle Traders	W435
Distinctive Wax Company	M556
Distinctive Wax Company	R537
Distinctive Wax Company	W435
Enhanced Eggbeater Corporation	R537
First-Rate Glass Corporation	M556

Figura 12.6
Incluindo duas colunas no intervalo de saída de uma consulta de Valores Únicos obtemos todas as combinações de Cliente e Produto.

Utilizando Filtro Avançado com critérios como intervalos

Como o nome sugere, o Filtro Avançado é normalmente utilizado para filtrar registros — em outras palavras, para obter um subconjunto de dados. Você especifica o subconjunto configurando um intervalo de critérios. Mesmo se estiver familiarizado com os critérios, não deixe de utilizar a poderosa fórmula Booleana em intervalos de critérios, que apresentaremos mais adiante neste capítulo, na seção “Os critérios mais complexos — substituindo a lista de valores por uma condição criada como o resultado de uma fórmula”.

Configure um intervalo de critérios em uma área em branco da planilha. Um intervalo de critérios sempre inclui duas ou mais linhas. A primeira linha do intervalo de critérios contém um ou mais valores de cabeçalho de campo para corresponder àquele(s) no intervalo de dados que você quer filtrar. A segunda linha contém um valor que mostra os registros que serão extraídos. Na Figura 12.8, o intervalo J1:J2 é o intervalo de critérios e L1 é o intervalo de saída.

Na interface com o usuário do Excel, para extrair uma lista única de produtos que foram comprados por um determinado cliente, selecione Filtro Avançado e configure a caixa de diálogo Filtro Avançado, como mostrado anteriormente na Figura 12.7. A Figura 12.8 mostra os resultados.

Figura 12.7
Isso se refere ao critério mais simples que existe; para obter uma lista única de produtos comprados pela empresa Cool Saddle Traders, configure o intervalo de critérios mostrado em J1:J2.

J	K	L	M
Cliente		Produto	
Cool Saddle Traders		R537	
		M556	
		W435	

Figura 12.8
Os resultados do filtro avançado que utiliza o intervalo de critérios e busca uma lista única de produtos. Com certeza, critérios mais complexos e interessantes podem ser criados.

No VBA, você utiliza o seguinte código para realizar um filtro avançado equivalente:

```
Sub UniqueProductsOneCustomer()
    Dim IRange As Range
    Dim ORange As Range
    Dim CRange As Range

    ' Localiza o tamanho do conjunto de dados de hoje
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    NextCol = Cells(1, Columns.Count).End(xlToLeft).Column + 2

    ' Configura o intervalo de saída com um cliente
    Cells(1, NextCol).Value = Range("D1").Value
    ' Na realidade, esse valor deve ser passado a partir do userform
    Cells(2, NextCol).Value = Range("D2").Value
    Set CRange = Cells(1, NextCol).Resize(2, 1)

    ' Configura intervalo de resultados. Copia o título de B1 aí
    Range("B1").Copy Destination:=Cells(1, NextCol + 2)
    Set ORange = Cells(1, NextCol + 2)

    ' Define o intervalo de entrada
    Set IRange = Range("A1").Resize(FinalRow, NextCol - 2)

    ' Faz o filtro avançado obter a lista única de clientes & produto
    IRange.AdvancedFilter Action:=xlFilterCopy, _
        CriteriaRange:=CRange, CopyToRange:=ORange, Unique:=True
    ' O que citamos anteriormente também poderia ser escrito como:
    'IRange.AdvancedFilter xlFilterCopy, CRange, ORange, True

    ' Determina quantas linhas únicas temos
    LastRow = Cells(Rows.Count, NextCol + 2).End(xlUp).Row

    ' Classifica os dados
    Cells(1, NextCol + 2).Resize(LastRow, 1).Sort Key1:=Cells(1, NextCol + 2), _
        Order1:=xlAscending, Header:=xlYes
End Sub
```

Unindo vários critérios com um OU lógico

Você pode querer filtrar registros que correspondam a um critério ou outro (por exemplo, extrair clientes que compraram o produto M556 ou R537). Isso é chamado critério OU lógico.

Quando o critério deve ser unido por um OU lógico, coloque-o em linhas subseqüentes do intervalo de critérios. Por exemplo, o intervalo de critérios mostrado em J1:J3, na Figura 12.9, informa que os clientes encomendaram o produto M556 ou o produto R537.

Figura 12.9
Coloque o critério em linhas sucessivas para uni-los com um OU. Esse intervalo de critérios obtém os clientes que encomendaram o produto M556 ou R537.

Unindo dois critérios com um E lógico

Em outras ocasiões, você desejará filtrar registros correspondentes a um e outro critério. Por exemplo, talvez queira extrair registros em que o produto vendido foi o W435 e a região era a Oeste. Isso é chamado de E lógico.

Para unir dois critérios por E, coloque ambos na mesma linha do intervalo de critérios. Por exemplo, o intervalo de critérios mostrado em J1:K2 da Figura 12.10 obtém clientes que encomendaram o produto W435 na região Oeste.

Figura 12.10
Coloque o critério na mesma linha para uni-los com E. O intervalo de critérios em J1:K2 obtém os clientes da região Oeste que fizeram pedido do produto W435.

Outros critérios como intervalos ligeiramente complexos

O intervalo de critérios mostrado na Figura 12.11 é baseado em dois campos diferentes. Eles são unidos com OU. A consulta localiza todos os registros da região Oeste ou registros em que o produto é o W435.

Figura 12.11
O intervalo de critérios em J1:K3 retorna registros nos quais a região é Oeste ou o produto é W435.

Pode ser útil unir dois critérios com OU onde a nova legislação da Califórnia influencia as remessas feitas para a Califórnia ou os produtos vindos de fábricas da Califórnia.

Os critérios mais complexos — substituindo a lista de valores por uma condição criada como o resultado de uma fórmula

É possível agrupar um intervalo de critérios com vários critérios E lógico e OU lógico. Embora isso possa funcionar em algumas situações, em outros cenários foge rapidamente do controle. Felizmente, o Excel considera os critérios em que os registros são selecionados como o resultado de uma fórmula para tratar essa situação.

Estudo de caso

Trabalhando com critérios muito complexos

Seus clientes gostaram tanto do relatório "Crie um cliente" que contrataram você para escrever um novo relatório. Nesse caso, eles poderiam selecionar qualquer cliente, produto, região ou uma combinação deles. Você pode adaptar com rapidez o userform frmReport para mostrar três caixas de listagem, como se vê na Figura 12.12.

No primeiro teste, imagine que você seleciona dois clientes e dois produtos. Nesse caso, seu programa terá de criar um intervalo de cinco linhas de critérios, como mostrado na Figura 12.13. Isso não é tão ruim.

É enlouquecedor quando alguém seleciona 10 produtos, todas as regiões, exceto a região doméstica, e todos os clientes, exceto o cliente interno. Seu intervalo de critérios precisaria de combinações únicas dos campos selecionados. Isso poderia facilmente ser 10 produtos vezes 9 regiões vezes 499 clientes, ou mais de 44 mil linhas de intervalo de critérios. Você pode rapidamente acabar com um intervalo de critérios que abrange milhares de linhas e três colunas. Uma vez fui tolo o bastante por realmente tentar executar um filtro avançado com um intervalo de critérios assim. Esse intervalo ainda estaria tentando calcular se eu não tivesse reinicializado o computador.

A solução para esse relatório é substituir as listas de valores por uma condição baseada em fórmula.

Figura 12.12
Essa forma superflexível permite que os clientes façam qualquer tipo de relatório que eles puderem imaginar. Isso cria alguns intervalos de critérios assustadores, a menos que você conheça a saída.

J	K
Cliente	Produto
Guarded Aerobic Corporation	R537
Improved Doghouse Traders	R537
Guarded Aerobic Corporation	W435
Improved Doghouse Traders	W435

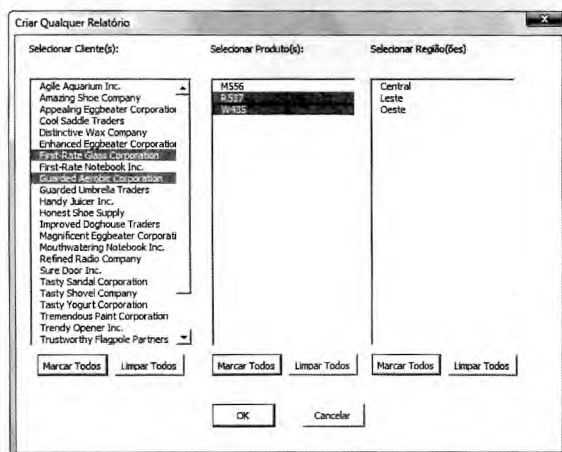



Figura 12.13
Esse intervalo de critérios retorna todos os registros nos quais dois clientes selecionados encomendaram algum dos dois produtos selecionados.

Configurando uma condição como o resultado de uma fórmula

Surpreendentemente, há uma versão muito obscura de critérios de filtro avançado que pode substituir o intervalo de critérios de 44 mil linhas no estudo de caso.

No modo alternativo de intervalo de critérios, a linha superior é deixada em branco. Não há nenhum título acima do critério. Os critérios configurados na Linha 2 são uma fórmula que resulta em `True` ou `False`. Se a fórmula contiver qualquer referência relativa à Linha 2 do intervalo de entrada, o Excel compara essa fórmula com cada linha do intervalo de entrada, uma a uma.

Por exemplo, se quiséssemos que todos os registros em que a Porcentagem de Lucro Bruto estivesse abaixo de 53 por cento, a fórmula criada em J2 referenciaria o Lucro em H2 e a Renda em F2. Deixaríamos J1 em branco para informar o Excel de que estávamos utilizando um critério baseado em fórmula. A célula J2 conteria a fórmula `=(H2/F2)<0.53`. O intervalo de critérios para o filtro avançado seria especificado como `J1:J2`.

Como o Excel executa o filtro avançado, copia logicamente a fórmula e a aplica a todas as linhas do banco de dados. Em qualquer lugar que a fórmula for avaliada como `True`, o registro será incluído no intervalo de saída.

Isso é incrivelmente poderoso e executa de forma muito rápida. Você pode combinar múltiplas fórmulas em colunas ou linhas adjacentes para unir os critérios de fórmula com E ou OU, assim como faz com critérios comuns.

Estudo de caso

Utilizando Condições Baseadas em Fórmula na interface com o usuário do Excel

Você pode utilizar condições baseadas em fórmula para resolver facilmente o relatório introduzido no estudo de caso anterior.

Para ilustrar, à direita do intervalo de critérios, configure uma coluna de células com a lista de clientes selecionados. Atribua um nome ao intervalo, como `MyCust`. Na célula J2 do intervalo de critérios, insira uma fórmula como `=Not(ISNA(Match(D2, MyCust,False)))`.

À direita do intervalo MyCust, configure um intervalo com uma lista de produtos selecionados. Atribua o nome `MyProd` a esse intervalo. No K2 do intervalo de critérios, adicione uma fórmula para verificar produtos: `=NOT(ISNA(Match(B2,MyProd,False)))`.

À direita do intervalo `MyProd`, configure um intervalo com uma lista de regiões selecionadas. Atribua o nome `MyRegion` a esse intervalo. Na L2 do intervalo de critérios, adicione uma fórmula para verificar as regiões selecionadas: `=NOT(ISNA(Match(A2,MyRegion,False)))`.

Agora, com um intervalo de critérios de J1:L2, você pode recuperar efetivamente os registros que correspondem a qualquer combinação de seleção do userform.

Utilizando Condições Baseadas em Fórmula com o VBA

A seguir, o código para este novo userform. Note a lógica em `OKButton_Click` que cria a fórmula. A Figura 12.14 mostra a planilha Excel antes do Filtro Avançado ser executado:

```
Private Sub CancelButton_Click()
    Unload Me
End Sub

Private Sub cbSubAll_Click()
    For i = 0 To lbCust.ListCount - 1
        Me.lbCust.Selected(i) = True
    Next i
End Sub

Private Sub cbSubClear_Click()
    For i = 0 To lbCust.ListCount - 1
        Me.lbCust.Selected(i) = False
    Next i
End Sub

Private Sub CommandButton1_Click()
    ' Limpa todos os produtos
    For i = 0 To lbProduct.ListCount - 1
        Me.lbProduct.Selected(i) = False
    Next i
End Sub

Private Sub CommandButton2_Click()
    ' Marca todos os produtos
    For i = 0 To lbProduct.ListCount - 1
        Me.lbProduct.Selected(i) = True
    Next i
End Sub

Private Sub CommandButton3_Click()
    ' Limpa todas as regiões
    For i = 0 To lbRegion.ListCount - 1
        Me.lbRegion.Selected(i) = False
    Next i
End Sub

Private Sub CommandButton4_Click()
    ' Marca todas as regiões
    For i = 0 To lbRegion.ListCount - 1
        Me.lbRegion.Selected(i) = True
    Next i
End Sub

Private Sub OKButton_Click()
    Dim CRange As Range, IRange As Range, ORange As Range
    ' Constrói um critério complexo em que ANDS agrupa todas as escolhas
    NextCCol = 10
    NextTCol = 15
```

```
        For j = 1 To 3
            Select Case j
                Case 1
                    MyControl = "lbCust"
                    MyColumn = 4
                Case 2
                    MyControl = "lbProduct"
                    MyColumn = 2
                Case 3
                    MyControl = "lbRegion"
                    MyColumn = 1
            End Select
            NextRow = 2
            ' Verifica o que foi selecionado.
            For i = 0 To Me.Controls(MyControl).ListCount - 1
                If Me.Controls(MyControl).Selected(i) = True Then
                    Cells(NextRow, NextTCol).Value = _
                        Me.Controls(MyControl).List(i)
                    NextRow = NextRow + 1
                End If
            Next i
            ' Se algo foi selecionado, cria uma nova fórmula de critério
            If NextRow > 2 Then
                ' A referência à Linha 2 deve ser relativa para funcionar
                MyFormula = "=NOT(ISNA(MATCH(RC" & MyColumn & ",R2C" & NextTCol & _
                    ":R" & NextRow - 1 & "C" & NextTCol & ",False)))"
                Cells(2, NextCCol).FormulaR1C1 = MyFormula
                NextTCol = NextTCol + 1
                NextCCol = NextCCol + 1
            End If
        Next j
        Unload Me

        ' A Figura 12.14 mostra a planilha nesse ponto
        ' Se construirmos um critério, define o intervalo de critérios
        If NextCCol > 10 Then
            Set CRange = Range(Cells(1, 10), Cells(2, NextCCol - 1))
            Set IRange = Range("A1").CurrentRegion
            Set ORange = Cells(1, 20)
            IRange.AdvancedFilter xlFilterCopy, CRange, ORange

            ' Limpa o critério
            Cells(1, 10).Resize(1, 10).EntireColumn.Clear
        End If

        ' Nesse ponto, os registros correspondentes estão em T1

    End Sub

    Private Sub UserForm_Initialize()
        Dim IRange As Range
        Dim ORange As Range

        ' Localiza o tamanho do conjunto de dados de hoje
        FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
        NextCol = Cells(1, Columns.Count).End(xlToLeft).Column + 2

        ' Define o intervalo de entrada
        Set IRange = Range("A1").Resize(FinalRow, NextCol - 2)

        ' Configura o intervalo de saída para o cliente. Copia o título de D1
        Range("D1").Copy Destination:=Cells(1, NextCol)
        Set ORange = Cells(1, NextCol)

        ' Faz o filtro avançado obter a lista única de clientes
        IRange.AdvancedFilter Action:=xlFilterCopy, CriteriaRange:="", _
            CopyToRange:=ORange, Unique:=True

        ' Determina quantos clientes únicos temos
        LastRow = Cells(Rows.Count, NextCol).End(xlUp).Row
```

```
    ' Classifica os dados
    Cells(1, NextCol).Resize(LastRow, 1).Sort Key1:=Cells(1, NextCol), _
        Order1:=xlAscending, Header:=xlYes

    With Me.lbCust
        .RowSource = ""
        FinalRow = Cells(Rows.Count, 10).End(xlUp).Row
        For Each cell In Cells(2, NextCol).Resize(LastRow - 1, 1)
            .AddItem cell.Value
        Next cell
    End With

    ' Apaga a lista temporária de clientes
    Cells(1, NextCol).Resize(LastRow, 1).Clear

    ' Configura o intervalo de saída para o produto. Copia o título de D1
    Range("B1").Copy Destination:=Cells(1, NextCol)
    Set ORange = Cells(1, NextCol)

    ' Faz o filtro avançado obter a lista única de clientes
    IRange.AdvancedFilter Action:=xlFilterCopy, _
        CopyToRange:=ORange, Unique:=True

    ' Determina quantos clientes únicos temos
    LastRow = Cells(Rows.Count, NextCol).End(xlUp).Row

    ' Classifica os dados
    Cells(1, NextCol).Resize(LastRow, 1).Sort Key1:=Cells(1, NextCol), _
        Order1:=xlAscending, Header:=xlYes

    With Me.lbProduct
        .RowSource = ""
        FinalRow = Cells(Rows.Count, 10).End(xlUp).Row
        For Each cell In Cells(2, NextCol).Resize(LastRow - 1, 1)
            .AddItem cell.Value
        Next cell
    End With

    ' Apaga a lista temporária de clientes
    Cells(1, NextCol).Resize(LastRow, 1).Clear

    ' Configura o intervalo de saída para a região. Copia o título a partir de A1 aí
    Range("A1").Copy Destination:=Cells(1, NextCol)
    Set ORange = Cells(1, NextCol)

    ' Faz o filtro avançado obter a lista única de clientes
    ' A Figura 12.15 mostra o estado da planilha antes dessa linha
    IRange.AdvancedFilter Action:=xlFilterCopy, CopyToRange:=ORange, _
        Unique:=True

    ' Determina quantos clientes únicos temos
    LastRow = Cells(Rows.Count, NextCol).End(xlUp).Row

    ' Classifica os dados
    Cells(1, NextCol).Resize(LastRow, 1).Sort Key1:=Cells(1, NextCol), _
        Order1:=xlAscending, Header:=xlYes

    With Me.lbRegion
        .RowSource = ""
        FinalRow = Cells(Rows.Count, 10).End(xlUp).Row
        For Each cell In Cells(2, NextCol).Resize(LastRow - 1, 1)
            .AddItem cell.Value
        Next cell
    End With

    ' Apaga a lista temporária de clientes
    Cells(1, NextCol).Resize(LastRow, 1).Clear

End Sub
```

Figura 12.14
A planilha antes de a macro executar o filtro avançado.

J2 =NÃO(É.NÃO.DISP(CORRESP($D2;$O$2:$O$26;FALSO)))

	G	H	I	J	K	L	M	N	O	P	Q
1	Custo Vendas Lucro										
2	11242	11568		VERDADEIRO	FALSO	FALSO			Agile Aquarium Inc.	M556	Central
3	4659	5586							Amazing Shoe Company	W435	Oeste
4	1082	1175							Appealing Eggbeater Corporation		
5	5621	5619							Cool Saddle Traders		
6	4497	4655							Distinctive Wax Company		
7	8659	9893							Enhanced Eggbeater Corporation		
8	4497	4707							First-Rate Glass Corporation		
9	3727	3133							First-Rate Notebook Inc.		
10	3727	4729							Guarded Aerobic Corporation		
11	6745	7061							Guarded Umbrella Traders		
12	10824	10906							Handy Juicer Inc.		
13	7454	8962							Honest Shoe Supply		
14	10118	10897							Magnificent Eggbeater Corporation		
15	10118	11320							Mouthwatering Notebook Inc.		
16	10118	11347							Refined Radio Company		
17	2795	3472							Sure Door Inc.		
18	4497	4647							Tasty Sandal Corporation		
19	9317	9793							Tasty Shovel Company		
20	1124	1277							Tremendous Paint Corporation		
21	932	808							Trendy Opener Inc.		
22	4659	4686							Trustworthy Flagpole Partners		
23	5590	6038							Unusual Raft Company		
24	10118	11770							User-Friendly Luggage Corporation		
25	3247	2714							Wonderful Doorbell Corporation		
26	1082	960							Wonderful Electronics Company		
27	8385	9120									

A Figura 12.14 mostra a planilha antes de o método `AdvancedFilter` ser chamado. O usuário selecionou clientes, produtos e regiões. A macro criou tabelas temporárias nas colunas O, P, Q para mostrar os valores que o usuário selecionou. O intervalo de critérios é J1:L2. Essa fórmula de critérios em J2 verifica se o valor em `$D2` está na lista de clientes selecionados em O. As fórmulas em K2 e L2 comparam `$B2` com a coluna P e `$A2` com a coluna Q.

ATENÇÃO

A ajuda do Excel VBA diz que, se você não especificar um intervalo de critérios, nenhum critério será utilizado. Isso não é verdadeiro no Excel 2007 — se nenhum intervalo de critérios for especificado, o filtro avançado herdará o intervalo de critérios do filtro avançado anterior. Você deve incluir `CriteriaRange:=""` para limpar o valor anterior.

Utilizando Condições Baseadas em Fórmula para retornar registros acima da média

Os critérios de fórmula de Condições Baseadas em Fórmula são excelentes, mas constituem um recurso raras vezes utilizado em uma função pouco utilizada. Algumas aplicações interessantes de negócios usam essa técnica. Por exemplo, essa fórmula de critério localizaria todas as linhas acima da média no conjunto de dados:

```
=$A2>Average($A$2:$A$60000)
```

Utilizando Filtrar no Local no Filtro Avançado

É possível filtrar um grande conjunto de dados no local. Nesse caso, você não precisa de um intervalo de saída. Normalmente, você especificaria o intervalo de critérios — caso contrário, retornaria 100 por cento dos registros e não haveria necessidade de fazer o filtro avançado!

Na interface com o usuário do Excel, faz sentido executar Filtrar no Local: você pode examinar atenta e facilmente a lista de filtros à procura de algo específico.

Executar Filtrar no Local no VBA é um pouco menos conveniente. A única maneira boa de examinar programaticamente os registros filtrados é utilizar a opção `xlCellTypeVisible` do método `SpecialCells`. Na interface com o usuário do Excel, a ação equivalente é selecionar Localizar & Selecionar, Ir Para Especial a partir da Faixa Início. No caixa de diálogo Ir Para Especial, selecione Somente Células Visíveis, como mostrado na Figura 12.15.

Figura 12.15
A opção Filtrar no Local oculta linhas que não correspondem ao critério selecionado, mas a única maneira de ver programaticamente os registros correspondentes é fazendo o equivalente a Selecionar Somente Células Visíveis na caixa de diálogo Ir Para Especial.

Para executar Filtrar no Local, utilize a *constante* `XLFilterInPlace` como o parâmetro Ação no comando `AdvancedFilter` e remova `CopyToRange` do comando:

```
IRange.AdvancedFilter Action:=xlFilterInPlace, CriteriaRange:=CRange, _
    Unique:=False
```

Então, o equivalente programático para fazer loop por Somente Células Visíveis é este código:

```
For Each cell In Range("A2:A" & FinalRow).SpecialCells(xlCellTypeVisible)
    Ctr = Ctr + 1
Next cell
MsgBox Ctr & " as células correspondem aos critérios"
```

Não capturando nenhum registro ao utilizar Filtrar no Local

Assim como Copiar Para, ao utilizar Filtrar no Local você tem de contar com a possibilidade de não ter nenhum registro que corresponda aos critérios. Nesse caso, porém, é mais difícil saber que nada foi retornado. Em geral, você descobre isso quando o método `SpecialCells` retorna um erro de tempo de execução 1004 — nenhuma célula foi localizada.

Para capturar essa condição, você tem de configurar uma interceptação de erro para antecipar o erro 1004 com o método `SpecialCells`. (Consulte o Capítulo 25, "Tratando erros", para obter mais informações sobre como capturar erros.)

```
    On Error GoTo NoRecs
    For Each cell In Range("A2:A" & FinalRow).SpecialCells(xlCellTypeVisible)
        Ctr = Ctr + 1
    Next cell
    On Error GoTo 0
    MsgBox Ctr & " as células correspondem aos critérios"
    Exit Sub
NoRecs:
    MsgBox "Nenhum registro corresponde aos critérios"
End Sub
```

Essa interceptação de erro funciona porque excluí especificamente a linha de cabeçalho do intervalo SpecialCells. A linha de cabeçalho é sempre visível depois de um filtro avançado. Incluí-la no intervalo impediria a ocorrência do erro 1004.

Mostrando todos os registros depois de Filtrar no Local

Depois de Filtrar no Local, você pode exibir todos os registros novamente utilizando o método `ShowAllData`:

```
ActiveSheet.ShowAllData
```

Utilizando Filtrar no Local com Somente Registros Exclusivos

É possível utilizar Filtrar no Local e Somente Registros Exclusivos. Entretanto, quando você especificou um intervalo de saída com apenas Produto e Cliente, o filtro avançado conseguiu lhe fornecer apenas as combinações únicas de Cliente e Produto. Se você pedir os registros únicos de um conjunto de dados com 10 campos, os únicos registros que não serão mostrados serão aqueles em que todos os 10 campos forem duplicatas exatas.

Cavalo de força real: `xlFilterCopy` com todos os registros em vez de Somente Registros Exclusivos

Os exemplos no começo deste capítulo discutiram como utilizar `xlFilterCopy` para obter uma lista de valores únicos em um campo. Utilizamos listas únicas de cliente, região e produto para preencher as caixas de listagem em nossos userforms específicos de relatório.

Um cenário mais comum, porém, é utilizar um filtro avançado para retornar todos os registros que correspondam ao critério. Depois de o usuário selecionar o cliente para o qual fazer relatório, um filtro avançado pode extrair todos os registros desse cliente.

Em todos os exemplos nas seções a seguir, você quer deixar a caixa de seleção Somente registros exclusivos desmarcada. Você faz isso no VBA especificando `Unique:=False` como um parâmetro para o método `AdvancedFilter`.

É fácil fazer isso e existem algumas opções poderosas. Se precisar apenas de um subconjunto de campos para um relatório, copie somente os títulos de campo para o intervalo de saída. Se quiser seqüenciar novamente os campos para que apareçam exatamente da maneira como você precisa que sejam exibidos no relatório, pode fazer isso alterando a seqüência dos títulos no intervalo de saída.

Apresento três exemplos rápidos, passo a passo, para mostrar as opções disponíveis.

Copiando todas as colunas

Para copiar todas as colunas, especifique uma única célula em branco como o intervalo de saída. Você obterá todas as colunas desses registros que correspondem aos critérios, conforme mostrado na Figura 12.16:

```
Sub AllColumnsOneCustomer()
    Dim IRange As Range
    Dim ORange As Range
    Dim CRange As Range

    ' Localiza o tamanho do conjunto de dados de hoje
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    NextCol = Cells(1, Columns.Count).End(xlToLeft).Column + 2

    ' Configura o intervalo de critérios com um cliente
    Cells(1, NextCol).Value = Range("D1").Value
    ' Na realidade, esse valor deve ser passado a partir do userform
    Cells(2, NextCol).Value = Range("D2").Value
    Set CRange = Cells(1, NextCol).Resize(2, 1)

    ' Configura intervalo de resultados. É uma única célula em branco
    Set ORange = Cells(1, NextCol + 2)

    ' Define o intervalo de entrada
    Set IRange = Range("A1").Resize(FinalRow, NextCol - 2)

    ' Faz o filtro avançado obter a lista única de clientes & produtos
    IRange.AdvancedFilter Action:=xlFilterCopy, _
        CriteriaRange:=CRange, CopyToRange:=ORange

End Sub
```

J	K	L	M	N	O	P	Q	R	S
Cliente		Região	Produto	Data	Cliente	Quantidade	Receita	Custo Vendas	Lucro
Trustworthy Flagpole		Leste	R537	24-jul-07	Trustworth	1000	22810	11242	11568
		Leste	W435	8-set-07	Trustworth	200	4742	2165	2577
		Oeste	M556	12-set-07	Trustworth	300	5700	2795	2905
		Central	W435	14-set-07	Trustworth	600	12282	6494	5788
		Leste	R537	17-set-07	Trustworth	100	2257	1082	1175
		Central	R537	18-set-07	Trustworth	1000	22680	11242	11438
		Leste	W435	18-set-07	Trustworth	600	13206	6494	6712
		Central	M556	19-set-07	Trustworth	900	16209	8385	7824
		Central	M556	26-set-07	Trustworth	200	4010	1863	2147
		Leste	W435	6-out-07	Trustworth	200	4526	2165	2361
		Oeste	W435	10-out-07	Trustworth	100	2157	1082	1075

Figura 12.16
Ao utilizar `xlFilterCopy` com um intervalo de saída em branco, você obtém todas colunas na mesma ordem em que aparecem no intervalo de lista original.

Copiando um subconjunto de colunas e reordenando

Se você estiver fazendo o filtro avançado para enviar registros para um relatório, é possível que só precise de um subconjunto de colunas e de que essas colunas estejam em uma seqüência diferente.

Eis um exemplo que concluirá o exemplo frmReport do início do capítulo. Como você se lembra, frmReport permite que o usuário do sistema selecione um cliente. O botão OK deve chamar a rotina `RunCustReport`, que passa um parâmetro para identificar o cliente para o qual será preparado um relatório.

Imagine que esse é um relatório que será enviado para o cliente. De fato, o cliente não se preocupa com a região vizinha e nós, definitivamente, não queremos revelar o custo das mercadorias vendidas ou o nosso lucro.

Supondo que colocaremos o cliente do nosso usuário no título do relatório, os campos que realmente precisamos para produzir o relatório são Data, Quantidade, Produto, Receita.

O seguinte código copia esses títulos para o intervalo de saída. O filtro avançado produz os dados, conforme mostrado na Figura 12.17. O programa então continua a copiar os registros correspondentes para uma nova pasta de trabalho. Um título e a linha total são adicionados e o relatório é salvo com o nome do cliente. A Figura 12.18 mostra o relatório final.

```
Sub RunCustReport(WhichCust As Variant)
    Dim IRange As Range
    Dim ORange As Range
    Dim CRange As Range
    Dim WBN As Workbook
    Dim WSN As Worksheet
    Dim WSO As Worksheet

    Set WSO = ActiveSheet
```

```
    ' Localiza o tamanho do conjunto de dados de hoje
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    NextCol = Cells(1, Columns.Count).End(xlToLeft).Column + 2

    ' Configura o intervalo de critérios com um único cliente
    Cells(1, NextCol).Value = Range("D1").Value
    Cells(2, NextCol).Value = WhichCust
    Set CRange = Cells(1, NextCol).Resize(2, 1)

    ' Configura o intervalo de resultados. Queremos Data, Quantidade, Produto, Receita
    ' Essas colunas estão em C, E, B e F
    Cells(1, NextCol + 2).Resize(1, 4).Value = _
        Array(Cells(1, 3), Cells(1, 5), Cells(1, 2), Cells(1, 6))
    Set ORange = Cells(1, NextCol + 2).Resize(1, 4)

    ' Define o intervalo de entrada
    Set IRange = Range("A1").Resize(FinalRow, NextCol - 2)

    ' Faz o filtro avançado obter a lista única de clientes & produtos
    IRange.AdvancedFilter Action:=xlFilterCopy, _
        CriteriaRange:=CRange, CopyToRange:=ORange

    ' Nesse ponto, os dados são semelhantes à Figura 12.18

    ' Cria uma nova pasta de trabalho com uma planilha em branco para armazenar a saída
    Set WBN = Workbooks.Add(xlWBATWorksheet)
    Set WSN = WBN.Worksheets(1)

    ' Configura um título em WSN
    WSN.Cells(1, 1).Value = "Relatório de vendas para " & WhichCust

    ' Copia os dados de WSO para WSN
    WSO.Cells(1, NextCol + 2).CurrentRegion.Copy Destination:=WSN.Cells(3, 1)
    TotalRow = WSN.Cells(Rows.Count, 1).End(xlUp).Row + 1
    WSN.Cells(TotalRow, 1).Value = "Total"
    WSN.Cells(TotalRow, 2).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
    WSN.Cells(TotalRow, 4).FormulaR1C1 = "=SUM(R2C:R[-1]C)"

    ' Formata o novo relatório com negrito
    WSN.Cells(3, 1).Resize(1, 4).Font.Bold = True
    WSN.Cells(TotalRow, 1).Resize(1, 4).Font.Bold = True
    WSN.Cells(1, 1).Font.Size = 18

    WBN.SaveAs "C:\" & WhichCust & ".xls"
    WBN.Close SaveChanges:=False

    WSO.Select

    ' Limpa o intervalo de saída, etc.
    Range("J1:Z1").EntireColumn.Clear

End Sub
```

Figura 12.17
Imediatamente depois do filtro avançado, temos apenas as colunas e registros necessários para o relatório.

J	K	L	M	N	O
Cliente		Data	Quantidade	Produto	Receita
Cool Saddle Traders		27-jul-07	400	R537	9152
		30-jul-07	600	R537	13806
		21-ago-07	400	M556	7136
		28-set-07	100	R537	2358
		4-out-07	100	R537	1819
		26-out-07	100	R537	2484
		7-mar-08	200	W435	4270
		22-ago-08	700	W435	12145

Figura 12.18
Depois de copiar os dados filtrados para uma nova planilha e aplicar alguma formatação, temos um relatório bem diagramado para enviar a cada cliente.

	A	B	C	D	E	F
1	Relatório de Vendas para Cool Saddle Traders					
2						
3	**Data**	**Quantidade**	**Produto**	**Receita**		
4	27-jul-07	400	R537	9152		
5	30-jul-07	600	R537	13806		
6	21-ago-07	400	M556	7136		
7	28-set-07	100	R537	2358		
8	4-out-07	100	R537	1819		
9	26-out-07	100	R537	2484		
10	7-mar-08	200	W435	4270		
11	22-ago-08	700	W435	12145		
12	**Total**	**2600**		**53170**		
13						

Estudo de caso

Utilizando dois tipos de filtros avançados para criar um relatório para cada cliente

O exemplo final de filtro avançado deste capítulo utiliza várias técnicas de filtro avançado. Digamos que, depois de importar o registro de fatura, você queira enviar um resumo de compra para cada cliente. O processo seria como o que mostramos a seguir:

1. Execute um filtro avançado para solicitar valores únicos a fim de obter uma lista de clientes em J. Esse `AdvancedFilter` especificaria o parâmetro `Unique:=True` e utilizaria `CopyToRange` que inclui um único título para Cliente:

```
'Configura o intervalo de resultados. Copia o título de D1
Range("D1").Copy Destination:=Cells(1, NextCol)
Set ORange = Cells(1, NextCol)

'Define o intervalo de entrada
Set IRange = Range("A1").Resize(FinalRow, NextCol - 2)

'Faz o filtro avançado obter a lista única de clientes
IRange.AdvancedFilter Action:=xlFilterCopy, CriteriaRange:="", _
    CopyToRange:=ORange, Unique:=True
```

2. Para cada cliente na lista de clientes únicos na coluna J, realize os passos de 3 a 7. Localize o número de clientes no intervalo de saída no passo 1. Em seguida, utilize um loop `For Each Cell` para iterar pelos clientes:

```
'Faz loop por cada cliente
FinalCust = Cells(Rows.Count, NextCol).End(xlUp).Row
For Each cell In Cells(2, NextCol).Resize(FinalCust - 1, 1)
    ThisCust = cell.Value
    ' ... Passos 3 a 7 aqui
Next Cell
```

3. Crie um intervalo de critérios em L1:L2 para ser utilizado em um novo filtro avançado. O intervalo de critérios incluiria o título 'Cliente' em L1 e o nome do cliente dessa iteração do loop na célula L2:

```
' Configura o intervalo de critérios com um único cliente
Cells(1, NextCol + 2).Value = Range("D1").Value
Cells(2, NextCol + 2).Value = ThisCust
Set CRange = Cells(1, NextCol + 2).Resize(2, 1)
```

4. Faça um filtro avançado copiar registros correspondentes desse cliente para a Coluna N. Essa instrução `Advanced Filter` especificaria o parâmetro `Unique:=False`. Como queremos apenas as colunas de Data, Quantidade, Produto e Receita, `CopyToRange` especifica um intervalo de quatros colunas com esses títulos copiados na ordem adequada:

```
' Configura intervalo de resultados. Queremos Data, Quantidade, Produto, Renda
' Essas colunas estão em C, E, B e F
Cells(1, NextCol + 4).Resize(1, 4).Value = _
    Array(Cells(1, 3), Cells(1, 5), Cells(1, 2), Cells(1, 6))
Set ORange = Cells(1, NextCol + 4).Resize(1, 4)

' Faz o filtro avançado obter a lista única de clientes & produtos
IRange.AdvancedFilter Action:=xlFilterCopy, CriteriaRange:=CRange, _
    CopyToRange:=Orange
```

5. Copie os registros de cliente para uma planilha de relatório em uma nova pasta de trabalho. O código VBA utiliza o método `Workbooks.Add` para criar uma nova pasta de trabalho em branco. Os registros extraídos do passo 4 são copiados para a célula A3 da nova pasta de trabalho:

```
' Cria uma nova pasta de trabalho com uma planilha em branco para armazenar a saída
Set WBN = Workbooks.Add(xlWBATWorksheet)
Set WSN = WBN.Worksheets(1)

' Copia os dados de WSO para WSN
WSO.Cells(1, NextCol + 4).CurrentRegion.Copy _
    Destination:=WSN.Cells(3, 1)
```

6. Formate o relatório com um título e totais. No VBA, adicione um título que expresse o nome do cliente na célula A1. Aplique negrito aos títulos e adicione um total abaixo da linha final:

```
' Configura um título em WSN
WSN.Cells(1, 1).Value = "Relatório de vendas para " & ThisCust
```

```
    TotalRow = WSN.Cells(Rows.Count, 1).End(xlUp).Row + 1
    WSN.Cells(TotalRow, 1).Value = "Total"
    WSN.Cells(TotalRow, 2).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
    WSN.Cells(TotalRow, 4).FormulaR1C1 = "=SUM(R2C:R[-1]C)"

    ' Formata o novo relatório com negrito
    WSN.Cells(3, 1).Resize(1, 4).Font.Bold = True
    WSN.Cells(TotalRow, 1).Resize(1, 4).Font.Bold = True
    WSN.Cells(1, 1).Font.Size = 18
```

7. Utilize `SaveAs` para salvar a pasta de trabalho baseada no nome do cliente. Depois de salvá-la, feche a nova pasta de trabalho. Retorne à pasta de trabalho original e limpe o intervalo de saída para preparar para a próxima passagem pelo loop:

```
    WBN.SaveAs "C:\Reports\" & ThisCust & ".xls"
    WBN.Close SaveChanges:=False

    WSO.Select
    Set WSN = Nothing
    Set WBN = Nothing

    ' Limpa o intervalo de saída, etc.
    Cells(1, NextCol + 2).Resize(1, 10).EntireColumn.Clear
```

A seguir, o código completo:

```
Sub RunReportForEachCustomer()
    Dim IRange As Range
    Dim ORange As Range
    Dim CRange As Range
    Dim WBN As Workbook
    Dim WSN As Worksheet
    Dim WSO As Worksheet

    Set WSO = ActiveSheet
    ' Localiza o tamanho do conjunto de dados de hoje
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    NextCol = Cells(1, Columns.Count).End(xlToLeft).Column + 2

    ' Primeiro obtém uma lista única de clientes em J
    ' Configura o intervalo de resultados. Copia o título de D1
    Range("D1").Copy Destination:=Cells(1, NextCol)
    Set ORange = Cells(1, NextCol)

    ' Define o intervalo de entrada
    Set IRange = Range("A1").Resize(FinalRow, NextCol - 2)

    ' Faz o filtro avançado obter a lista única de clientes
    IRange.AdvancedFilter Action:=xlFilterCopy, CriteriaRange:="", _
        CopyToRange:=ORange, Unique:=True

    ' Faz loop por cada cliente
    FinalCust = Cells(Rows.Count, NextCol).End(xlUp).Row
    For Each cell In Cells(2, NextCol).Resize(FinalCust - 1, 1)
        ThisCust = cell.Value

        ' Configura o critério de intervalo com um único cliente
        Cells(1, NextCol + 2).Value = Range("D1").Value
        Cells(2, NextCol + 2).Value = ThisCust
        Set CRange = Cells(1, NextCol + 2).Resize(2, 1)

        ' Configura o intervalo de resultados. Queremos Data, Quantidade, Produto, Receita
        ' Essas colunas estão em C, E, B e F
        Cells(1, NextCol + 4).Resize(1, 4).Value = _
            Array(Cells(1, 3), Cells(1, 5), Cells(1, 2), Cells(1, 6))
        Set ORange = Cells(1, NextCol + 4).Resize(1, 4)

        ' Faz o filtro avançado obter a lista única de clientes & produto
        IRange.AdvancedFilter Action:=xlFilterCopy, CriteriaRange:=CRange, _
            CopyToRange:=ORange

        ' Cria uma nova pasta de trabalho com uma planilha em branco para armazenar a saída
        Set WBN = Workbooks.Add(xlWBATWorksheet)
```

```
        Set WSN = WBN.Worksheets(1)

        ' Copia os dados de WSO para WSN
        WSO.Cells(1, NextCol + 4).CurrentRegion.Copy _
            Destination:=WSN.Cells(3, 1)

        ' Configura um título em WSN
        WSN.Cells(1, 1).Value = "Relatório de vendas para " & ThisCust

        TotalRow = WSN.Cells(Rows.Count, 1).End(xlUp).Row + 1
        WSN.Cells(TotalRow, 1).Value = "Total"
        WSN.Cells(TotalRow, 2).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
        WSN.Cells(TotalRow, 4).FormulaR1C1 = "=SUM(R2C:R[-1]C)"

        ' Formata o novo relatório com negrito
        WSN.Cells(3, 1).Resize(1, 4).Font.Bold = True
        WSN.Cells(TotalRow, 1).Resize(1, 4).Font.Bold = True
        WSN.Cells(1, 1).Font.Size = 18

        WBN.SaveAs "C:\Reports\" & ThisCust & ".xlsx"
        WBN.Close SaveChanges:=False

        WSO.Select
        Set WSN = Nothing
        Set WBN = Nothing

        ' Limpa o intervalo de saída, etc.
        Cells(1, NextCol + 2).Resize(1, 10).EntireColumn.Clear
    Next cell

    Cells(1, NextCol).EntireColumn.Clear
    MsgBox FinalCust - 1 & " Os relatórios foram criados!"
End Sub
```

Isso é um código notável de 75 linhas. Ao incorporar dois filtros avançados e não muito mais, conseguimos produzir uma ferramenta que criou 27 relatórios em menos de um minuto (veja Figura 12.19). Até mesmo um usuário avançado do Excel levaria de 2 a 3 minutos para criar cada um desses relatórios manualmente. Em menos de 60 segundos, esse código facilmente poupará algumas horas de alguém toda vez que houver a necessidade de criar esses relatórios. Imagine um cenário real em que há centenas de clientes. Garanto que existem pessoas em todas as cidades do mundo criando esses relatórios manualmente no Excel, porque simplesmente não conhecem o poder do Excel VBA.

Figura 12.19
A produtividade de funcionários administrativos decolaria se eles soubessem criar 27 relatórios em menos de um minuto.

Nome	Tamanho	Tipo	Modificado em
Agile Aquarium Inc..xls	9 KB	Planilha do Microsoft Of...	21/08/2008 15:50
Amazing Shoe Company.xls	10 KB	Planilha do Microsoft Of...	21/08/2008 15:49
Appealing Eggbeater Corporation.xls	9 KB	Planilha do Microsoft Of...	21/08/2008 15:50
Cool Saddle Traders.xls	9 KB	Planilha do Microsoft Of...	21/08/2008 15:49
Distinctive Wax Company.xls	11 KB	Planilha do Microsoft Of...	21/08/2008 15:50
Enhanced Eggbeater Corporation.xls	12 KB	Planilha do Microsoft Of...	21/08/2008 15:50
First-Rate Glass Corporation.xls	9 KB	Planilha do Microsoft Of...	21/08/2008 15:50
First-Rate Notebook Inc..xls	9 KB	Planilha do Microsoft Of...	21/08/2008 15:50
Guarded Aerobic Corporation.xls	12 KB	Planilha do Microsoft Of...	21/08/2008 15:50
Guarded Umbrella Traders.xls	11 KB	Planilha do Microsoft Of...	21/08/2008 15:50
Handy Juicer Inc..xls	9 KB	Planilha do Microsoft Of...	21/08/2008 15:50
Honest Shoe Supply.xls	9 KB	Planilha do Microsoft Of...	21/08/2008 15:50
Improved Doghouse Traders.xls	9 KB	Planilha do Microsoft Of...	21/08/2008 15:50
Magnificent Eggbeater Corporation.xls	12 KB	Planilha do Microsoft Of...	21/08/2008 15:50

Utilizando o AutoFiltro

O recurso AutoFiltro foi adicionado ao Excel porque as pessoas achavam muito difícil encontrar os filtros. Eles são muito bons quando utilizados na interface com o usuário do Excel. Raras vezes tive a oportunidade de utilizá-los no Excel VBA.

No Excel 2007, a Microsoft renomeou AutoFiltro como Filtro e adicionou vários filtros dinâmicos novos. Esses filtros permitem escolher registros com datas que caem no último trimestre, na próxima semana ou neste ano. Embora a interface com o usuário do Excel referencie-os como filtros, o código VBA ainda utiliza o termo AutoFiltro ao se referir a esses filtros.

A natureza do AutoFiltro é que o Excel sempre filtrará no local. Portanto, você tem de utilizar o método `SpecialCells (xlCellTypeVisible)` para acessar as linhas retornadas do filtro.

Ativando o AutoFiltro com o código

Há apenas um conjunto de dados autofiltrado em cada planilha. Para ativar o AutoFiltro, você aplica o método `AutoFilter` a cada célula no conjunto de dados. Por exemplo, o seguinte código ativa as listas suspensas AutoFiltro:

```
Range("A1").AutoFilter
```

O método `AutoFilter` é um alternador. Se as listas suspensas AutoFiltro já estiverem ativadas, executar o código anterior desativará a lista suspensa AutoFiltro. O Excel 2007 adiciona um nova propriedade `FilterMode`, mas ela só é configurada como `True` se alguém tiver selecionado um valor em uma lista suspensa AutoFiltro. Portanto, para descobrir se o AutoFiltro já está ativado, você poderia fazer as macros a seguir ativarem e/ou desativarem as listas suspensas AutoFiltro:

```
Sub TurnOnAutoFilter()
    ' Ativa AutoFiltros
    Worksheets("SalesReport").Select
    On Error Resume Next
    x = ActiveSheet.AutoFilter.Range.Areas.Count
    If Err.Number > 0 Then
        ActiveSheet.Range("A1").AutoFilter
    End If
    On Error Resume Next
End Sub
```

Utilize este código para desativar a lista suspensa AutoFiltro:

```
Sub TurnOffAutoFilter()
    ' Desativa AutoFiltros
    Worksheets("SalesReport").Select
    On Error Resume Next
    x = ActiveSheet.AutoFilter.Range.Areas.Count
    If Err.Number = 0 Then
        ActiveSheet.Range("A1").AutoFilter
    End If
    On Error Resume Next
End Sub
```

Desativando algumas listas suspensas no AutoFiltro

Um recurso excelente está disponível apenas no Excel VBA. Quando você usa o AutoFiltro em uma lista, na interface com o usuário do Excel, todas as colunas no conjunto de dados obtêm um campo suspenso na linha de título. Às vezes, você tem um campo que não faz muito sentido para o AutoFiltro. Por exemplo, em nosso conjunto de dados atual, talvez você queira fornecer as listas suspensas de AutoFiltro para Região, Produto e Cliente, mas não os campos numéricos ou de data. Depois de configurar o AutoFiltro, você precisa de uma linha de código para desativar cada lista suspensa que não quiser exibir. O código seguinte desativa a lista suspensa das colunas C, E, F, G e H:

```
Sub AutoFilterCustom()
    Range("A1").AutoFilter Field:=3, VisibleDropDown:=False
    Range("A1").AutoFilter Field:=5, VisibleDropDown:=False
    Range("A1").AutoFilter Field:=6, VisibleDropDown:=False
    Range("A1").AutoFilter Field:=7, VisibleDropDown:=False
    Range("A1").AutoFilter Field:=8, VisibleDropDown:=False
End Sub
```

Acho que usar essa ferramenta é um prazer relativamente raro. Na maioria das vezes, o Excel VBA permite que façamos coisas que seriam possíveis na interface com o usuário (embora nos deixe fazê-las muito rapidamente). O parâmetro `VisibleDropDown` realmente permite fazer algo no VBA que, em geral, não está disponível na interface com o usuário do Excel. Seus clientes quebrarão a cabeça para descobrir como configurar o excelente AutoFiltro com apenas algumas colunas filtráveis (veja Figura 12.20).

	A	B	C	D	E	F	G	H
1	Região	Produto	Data	Cliente	Quantidade	Receita	Custo Vendas	Lucro
2	Leste	R537	24-jul-07	Trustworthy Flagpole Partners	1000	22810	11242	11568
3	Leste	M556	25-jul-07	Amazing Shoe Company	500	10245	4659	5586
4	Central	W435	25-jul-07	Amazing Shoe Company	100	2257	1082	1175
5	Central	R537	26-jul-07	Mouthwatering Notebook Inc.	500	11240	5621	5619
6	Leste	R537	27-jul-07	Cool Saddle Traders	400	9152	4497	4655
7	Leste	W435	27-jul-07	Tasty Shovel Company	800	18552	8659	9893
8	Central	R537	27-jul-07	Mouthwatering Notebook Inc.	400	9204	4497	4707

Figura 12.20 Utilizando o VBA, você pode configurar um AutoFiltro em que apenas certas colunas têm a lista suspensa AutoFiltro.

Filtrando uma coluna com AutoFiltros

Na primeira encarnação de AutoFiltros, você tinha de especificar um número de coluna, um critério, um operador e um segundo critério. Como os AutoFiltros eram limitados a duas condições, isso tratava qualquer possível cenário de filtragem.

Por exemplo, o código seguinte faz a filtragem para mostrar registros para o cliente Agile Aquarium. Como Cliente é a quarta coluna no conjunto de dados, o número do Campo é 4:

```
Sub SimpleFilter()
    Worksheets("SalesReport").Select
    Range("A1").AutoFilter
    Range("A1").AutoFilter Field:=4, _
        Criteria1:="=Agile Aquarium Inc."
End Sub
```

Para limpar o filtro da coluna do cliente, utilize este código:

```
Sub SimpleFilter()
    Worksheets("SalesReport").Select
    Range("A1").AutoFilter
    Range("A1").AutoFilter Field:=4
End Sub
```

Em versões anteriores do Excel, você podia unir dois critérios com os operadores OU ou E. O código a seguir filtraria a coluna de cliente para um de dois clientes, unidos pelo operador OU:

```
Sub SimpleOrFilter()
    Worksheets("SalesReport").Select
    Range("A1").AutoFilter
    Range("A1").AutoFilter Field:=4, _
        Criteria1:="=Agile Aquarium Inc.", _
        Operator:=xlOr, Criteria2:="=Amazing Shoe Company"
End Sub
```

O seguinte código retorna todos os clientes que iniciavam com as letras *A* a *E*:

```
Sub SimpleAndFilter()
    Worksheets("SalesReport").Select
    Range("A1").AutoFilter
    Range("A1").AutoFilter Field:=4, _
        Criteria1:=">=A", _
        Operator:=xlAnd, Criteria2:="<=EZZ"
End Sub
```

Como o comando AutoFiltro se tornou mais flexível, a Microsoft continuou a utilizar os mesmos três parâmetros, ainda que eles não façam nenhum sentido. Por exemplo, o Excel permitirá filtrar um campo que pede os cinco itens principais ou a base de 8 por cento dos registros. Para utilizar esse tipo de filtro, você especifica `"5"` ou `"8"` como o argumento `Criteria1` e então especifica `xlTop10Items`, `xlTop10Percent`, `xlBottom10Items`, `xlBottom10Percent` como o operador. O código seguinte produz os dez registros principais:

```
Sub Top10Filter()
    ' Os 12 principais registros de renda
    Worksheets("SalesReport").Select
    Range("A1").AutoFilter
    Range("A1").AutoFilter Field:=6, _
        Criteria1:="12", _
        Operator:=xlTop10Items
End Sub
```

O Excel 2007 oferece várias opções novas de filtro. O Excel continua a forçar essas opções de filtro para que se ajustem ao antigo modelo de objeto, no qual o comando de filtro deve ajustar-se a um operador e a até dois campos de critérios.

Selecionando múltiplos valores de um filtro

Se você especificar três ou mais itens de uma lista suspensa de filtros, o Excel VBA utilizará o operador `xlFilterValues` e especificará a lista completa de itens selecionados em um array passado para `Criteria1`. O código seguinte seleciona cinco clientes específicos da lista suspensa em D1:

```
Sub MultiSelectFilter()
    ' Seleciona muitos clientes
    Worksheets("SalesReport").Select
    Range("A1").AutoFilter
    Range("A1").AutoFilter Field:=4, _
        Criteria1:=Array( _
            "Amazing Shoe Company", "Cool Saddle Traders", _
            "Enhanced Eggbeater Corporation", _
            "First-Rate Notebook Inc.", "Handy Juicer Inc."), _
        Operator:=xlFilterValues
End Sub
```

Selecionando um intervalo dinâmico de data usando AutoFiltros

Talvez os recursos mais poderosos nos filtros do Excel 2007 sejam os novos filtros dinâmicos. Esses filtros permitem escolher registros que estão acima da média ou com um campo de data para selecionar períodos virtuais como Próxima Semana ou Ano Passado.

Para utilizar um filtro dinâmico, especifique `xlFilterDynamic` como o operador e depois utilize um de 34 valores como `Criteria1`. O código seguinte localiza todas as datas do ano que vem:

```
Sub DynamicAutoFilter()
    Worksheets("SalesReport").Select
    Range("A1").AutoFilter
    Range("A1").AutoFilter Field:=3, _
        Criteria1:=xlFilterNextYear, _
        Operator:=xlFilterDynamic
End Sub
```

Os itens a seguir relacionam todas as opções dinâmicas de critérios de filtro. Especifique esses valores como `Criteria1` no método `AutoFilter`:

- **Critérios para valores** — Utilize `xlFilterAboveAverage` ou `xlFilterBelowAverage` para localizar todas as linhas que estão acima ou abaixo da média. Observe que em Lake Wobegon, utilizar `xlFilterBelowAverage` provavelmente não retornará nenhum registro.

- **Critérios para períodos futuros** — Utilize `xlFilterTomorrow`, `xlFilterNextWeek`, `xlFilterNextMonth`, `xlFilterNextQuarter` ou `xlFilterNextYear` para localizar linhas que são incluídas em um certo período futuro. Observe que a próxima semana começa no domingo e termina no sábado.

- **Critérios para períodos atuais** — Utilize `xlFilterToday`, `xlFilterThisWeek`, `xlFilterThisMonth`, `xlFilterThisQuarter` ou `xlFilterThisYear` para localizar linhas que caem dentro do período atual. O Excel utilizará o clock do sistema para localizar o dia atual.

- **Critérios para períodos passados** — Utilize `xlFilterYesterday`, `xlFilterLastWeek`, `xlFilterLastMonth`, `xlFilterLastQuarter`, `xlFilterLastYear` ou `xlFilterYearToDate` para localizar linhas que caíram dentro de um período anterior.

- **Critérios para trimestres específicos** — Utilize `xlFilterDatesInPeriodQuarter1`, `xlFilterDatesInPeriodQuarter2`, `xlFilterDatesInPeriodQuarter3` ou `xlFilterDatesInPeriodQuarter4` para filtrar por linhas que caem dentro de um trimestre específico. Observe que esses filtros não diferenciam com base em um ano. Se você solicitar o trimestre 1, você poderia obter registros deste janeiro, último fevereiro e o próximo março.

- **Critérios para meses específicos** — Utilize `xlFilterDatesInPeriodJanuary` a `xlFitlerDatesInPeriodDecember` para filtrar registros que caem em determinado mês. Como nos trimestres, o filtro não faz a filtragem por um determinado ano qualquer.

Infelizmente, você não pode combinar critérios. Talvez ache que é possível especificar `xlFilterDatesInPeriodJanuary` como `Criteria1` e `xlFilterDatesNextYear` como `Criteria2`. Embora esse seja um pensamento brilhante, a Microsoft não suporta essa sintaxe (ainda).

Filtragem baseada em cor ou ícone

Outro novo recurso no Excel 2007 é a capacidade de filtrar com base na cor da fonte, na cor de preenchimento da célula ou no ícone de formatação condicional.

Se é esperado que o conjunto de dados tenha um conjunto de ícones, você pode filtrar para mostrar somente registros com um determinado ícone utilizando o operador `xlFilterIcon`.

Para o critério, você tem de saber qual conjunto de ícones foi aplicado e que ícone está dentro do conjunto. Os conjuntos de ícones são identificados com a utilização dos nomes mostrados na Coluna A da Figura 12.21. Os itens variam de 1 a 5. O próximo código filtra a coluna Renda para mostrar as linhas que contêm uma seta apontando para cima no conjunto de ícones de 5 setas acinzentadas:

```
Sub FilterByIcon()
    Worksheets("SalesReport").Select
    Range("A1").AutoFilter
    Range("A1").AutoFilter Field:=6, _
        Criteria1:=ActiveWorkbook.IconSets(xl5ArrowsGray).Item(5), _
        Operator:=xlFilterIcon
End Sub
```

Figura 12.21
Para procurar determinado ícone, você precisa conhecer o conjunto de ícones da coluna A e o número de itens da Linha 1.

Para localizar registros que não têm nenhum ícone de formatação condicional, utilize um operador de `xlFilterNoIcon` e não especifique nenhum critério.

Para localizar registros que têm uma cor de preenchimento particular, utilize um operador de `xlFilterCellColor` e especifique um valor RGB particular como critério. Esse código localiza todas as células vermelhas na coluna F:

```
Sub FilterByFillColor()
    Worksheets("SalesReport").Select
    Range("A1").AutoFilter
    Range("A1").AutoFilter Field:=6, _
        Criteria1:=RGB(255, 0, 0), Operator:=xlFilterCellColor
End Sub
```

Para localizar registros que não têm nenhuma cor de preenchimento, utilize um operador de `xlFilterNoFill` e não especifique nenhum critério.

Para localizar registros que tenham uma determinada cor de fonte, utilize um operador de `xlFilterFontColor` e especifique um determinado valor RGB como o critério. Esse código localiza todas as células com uma fonte vermelha na coluna F:

```
Sub FilterByFontColor()
    Worksheets("SalesReport").Select
    Range("A1").AutoFilter
    Range("A1").AutoFilter Field:=6, _
        Criteria1:=RGB(255, 0, 0), Operator:=xlFilterFontColor
End Sub
```

Para localizar registros que não têm uma cor de fonte específica, utilize um operador de `xlFilterAutomaticFillColor` e não especifique nenhum critério.

Estudo de caso

Utilizando AutoFiltro para copiar todos os registros da próxima semana

Suponha que você tenha um banco de dados que mostre tarefas de manutenção e a data em que elas precisam ser realizadas. Toda sexta-feira, você quer executar uma macro que gere uma lista das tarefas a serem realizadas na próxima semana.

O novo filtro dinâmico AutoFiltro da Próxima Semana é perfeito para essa tarefa.

Como o AutoFiltro pode ser aplicado a uma única célula no conjunto de dados, não há necessidade de descobrir o número de linhas ou colunas no conjunto de dados. Você pode simplesmente aplicar o AutoFiltro a `Range("A1")`.

A constante de filtro dinâmico para a próxima semana é `xlFilterNextWeek`. O seguinte código configura um AutoFiltro na planilha WSO:

```
'Ativa o AutoFiltro
WSO.Range("A1").AutoFilter
'Procura datas na coluna C
Range("A1").AutoFilter Field:=3, _
    Criteria1:=xlFilterNextWeek, _
    Operator:=xlFilterDynamic
```

Depois de definir o AutoFiltro, você precisa copiar as células visíveis do intervalo filtrado. Novidade do Excel 2007: você pode referir-se à extensão dos dados filtrados utilizando `ActiveWorksheet.AutoFilter.Range`. Essa propriedade retorna um intervalo que inclui os títulos pela última linha dos dados filtrados. Observe que ela também inclui as linhas ocultas. Você pode utilizar a propriedade `SpecialCells` para pegar somente as linhas visíveis:

```
WSO.AutoFilter.Range.SpecialCells(xlCellTypeVisible).Copy _
    Destination:=WSN.Cells(3, 1)
```

Colocar os passos em macros semelhantes às macros de Filtro Avançado gera uma macro que tem algumas linhas a menos do que a de Filtro Avançado:

```
Sub AutoFilterNextWeekCopy()
    Dim WBN As Workbook
    Dim WSN As Worksheet
    Dim WSO As Worksheet

    Set WSO = ActiveSheet

    ' Ativa o AutoFiltro
    WSO.Range("A1").AutoFilter
    ' Procura datas na coluna C

    Range("A1").AutoFilter Field:=3, _
        Criteria1:=xlFilterNextWeek, _
        Operator:=xlFilterDynamic

    ' Cria uma nova pasta de trabalho com uma planilha em branco para armazenar a saída
    Set WBN = Workbooks.Add(xlWBATWorksheet)
    Set WSN = WBN.Worksheets(1)

   ' Configura um título em WSN
    WSN.Cells(1, 1).Value = "Projects Due Next Week"

    ' Copia os dados de WSO para WSN
    WSO.AutoFilter.Range.SpecialCells(xlCellTypeVisible).Copy _
        Destination:=WSN.Cells(3, 1)

    ' Formata o novo relatório com negrito
    WSN.Cells(3, 1).Resize(1, FinalCol).Font.Bold = True
    WSN.Cells(1, 1).Font.Size = 18

    ' Salva como uma pasta de trabalho compatível com a macro
    WBN.SaveAs "C:\NextWeek.xlsm", xlOpenXMLWorkbookMacroEnabled

    ' Desativa o AutoFiltro
    WSO.Range("A1").AutoFilter
End Sub
```

O processo de copiar as células visíveis adiciona certa complexidade ao processo do AutoFiltro. Entretanto, se você precisar utilizar um dos novos filtros dinâmicos disponíveis no Excel 2007, essa é, sem dúvida, uma maneira mais fácil de pegar um intervalo dinâmico de datas.

Próximos passos

Utilizando as técnicas deste capítulo, você tem muitas técnicas de criação de relatório disponíveis com o emprego da enigmática ferramenta Filtro Avançado. O Capítulo 13, "Utilizando o VBA para criar tabelas dinâmicas", apresenta o recurso mais poderoso do Excel: a tabela dinâmica. A combinação de Filtro Avançado e tabelas dinâmicas cria ferramentas para fazer relatórios que permitem aplicações surpreendentes.

Utilizando o VBA para criar tabelas dinâmicas

13

Introduzindo tabelas dinâmicas

As tabelas dinâmicas são as ferramentas mais poderosas que o Excel oferece. O conceito foi primeiro posto em prática pela Lotus com seu produto Improv.

Adoro as tabelas dinâmicas, porque realmente são uma forma rápida de resumir quantidades maciças de dados. Você pode utilizar a tabela dinâmica básica comum para produzir um resumo conciso em segundos. Mas há tantas versões de tabelas dinâmicas que elas podem ser as ferramentas preferidas para muitos diferentes usos. Você pode construir tabelas dinâmicas para atuar como o mecanismo de cálculo para criar relatórios por estoque, por estilo ou para localizar rapidamente os cinco maiores ou dez menores de qualquer coisa.

Não estou sugerindo que você utilize o VBA para construir tabelas dinâmicas para dar para seu usuário, mas que as utilize como um meio para atingir um propósito — utilize-as para obter um resumo de dados e empregar esse resumo da melhor maneira.

Entendendo as versões

As tabelas dinâmicas evoluíram. Elas foram introduzidas no Excel 5 e aperfeiçoadas no Excel 97. No Excel 2000, a criação de tabela dinâmica no VBA mudou drasticamente. Alguns novos parâmetros foram adicionados no Excel 2002. Novas propriedades, como `PivotFilters` e `TableStyle2`, foram adicionadas no Excel 2007. Portanto, você precisa ser extremamente cuidadoso ao escrever códigos no Excel 2007 que possam ser executados no Excel 2003, no Excel 2000 ou no Excel 97.

Alguns ajustes simples fazem o código 2003 executar no 2000, mas uma grande inspeção é necessária para fazer qualquer código executar no Excel 97. Como já faz mais de 10 anos do lançamento do Excel 97 (e como a Microsoft não oferece suporte a esse produto há mais de cinco anos), este capítulo se concentra apenas em como utilizar o método dinâmico de cache introduzido no Excel 2000.

Novo no Excel 2007

Embora o conceito básico de tabelas dinâmicas no Excel 2007 seja o mesmo no Excel 2003, vários recursos novos estão disponíveis nas tabelas dinâmicas do Excel 2007. A guia Design é inteiramente nova, incluindo os conceitos de subtotais na parte superior, as opções de layout de relatório, as linhas em branco e os novos estilos de tabela dinâmica (PivotTable). O Excel 2007 oferece filtros melhores do que os das versões anteriores. Também tornou a funcionalidade de expandir e recolher mais visível, incluindo botões na grade da tabela dinâmica. Cada recurso novo adiciona um ou mais métodos ou propriedades ao VBA.

Se espera compartilhar sua macro de tabela dinâmica com pessoas que têm versões anteriores do Excel, você precisa evitar esses métodos. Sua melhor aposta é abrir uma pasta de trabalho do Excel 2003 no modo de Compatibilidade e gravar a macro enquanto a pasta de trabalho estiver nesse modo. Se utilizar a macro apenas no Excel 2007 ou em versão posterior, você pode utilizar todos os recursos novos.

NESTE CAPÍTULO

A Tabela 13.1 mostra os *métodos novos no* Excel 2007. Se você gravar uma macro que utiliza esses métodos, não poderá compartilhar a macro com usuários do Excel 2003 ou de versões anteriores.

Tabela 13.1 Métodos novos no Excel 2007

Método	Descrição
`ClearAllFilters`	Limpa todos os filtros na tabela dinâmica.
`ClearTable`	Remove todos os campos da tabela dinâmica, mas mantém a tabela dinâmica intacta.
`ConvertToFormulas`	Converte uma tabela dinâmica em fórmulas de cubo. Esse método é válido somente para tabelas dinâmicas baseadas em origens de dados OLAP.
`DisplayAllMemberPropertiesInTooltip`	Equivalente a Opções, Exibir, Mostrar Propriedades em Dicas de Tela.
`RowAxisLayout`	Altera o layout para todos os campos na área de linha. Os valores válidos são `xlCompactRow`, `xlTabularRow` ou `xlOutlineRow`.
`SubtotalLocation`	Controla se os subtotais aparecerão na parte superior ou inferior de cada grupo. Argumentos válidos são `xlAtTop` ou `xlAtBottom`.

A Tabela 13.2 lista as propriedades que são novas no Excel 2007. Se você registrar uma macro que referencie essas propriedades, não poderá compartilhar a macro com usuários do Excel 2003 ou de versões anteriores.

Tabela 13.2 Novas propriedades no Excel 2007

Propriedade	Descrição
`ActiveFilters`	Indica os filtros ativos na tabela dinâmica; essa é uma propriedade de leitura.
`AllowMultipleFilters`	Indica se um campo dinâmico pode receber a aplicação de vários filtros ao mesmo tempo.
`CompactLayoutColumnHeader`	Especifica a legenda que é exibida no cabeçalho de coluna de uma tabela dinâmica quando no formulário compacto de layout de linha.
`CompactLayoutRowHeader`	Especifica a legenda que exibe no cabeçalho de linha de uma tabela dinâmica quando em formulário compacto de layout de linha.
`CompactRowIndent`	Indica o incremento de recuo para itens dinâmicos quando o formulário de layout de linha compacta está ativado.
`DisplayContextTooltips`	Controla se as Dicas de Tela são exibidas para células de tabela dinâmica.
`DisplayFieldCaptions`	Controla a exibição de botões de filtro e de legendas dinâmicas de campo para linhas e colunas na grade.
`DisplayMemberPropertyTooltips`	Controla a exibição de propriedades de membro em Dicas de Tela.
`FieldListSortAscending`	Controla a ordem de classificação de campos na Lista de Campo da Tabela Dinâmica. Quando essa propriedade for `True`, os campos serão classificados em ordem alfabética. Quando estiver configurada como `False`, os campos serão apresentados na mesma seqüência das colunas de origem de dados.
`InGridDropZones`	Controla se você pode arrastar e soltar campos sobre a grade. Mudar o layout da tabela dinâmica também muda essa propriedade. Alterar essa propriedade força o layout a voltar para um layout de tabela.
`LayoutRowDefault`	Especifica as configurações de layout para campos dinâmicos quando são adicionados à tabela dinâmica pela primeira vez. Os valores válidos são `xlCompactRow`, `xlTabularRow` ou `xlOutlineRow`.
`PivotColumnAxis`	Retorna um objeto `PivotAxis` que representa todo o eixo de coluna.
`PivotRowAxis`	Retorna um `PivotAxis` objeto que representa todo o eixo de linha.
`PrintDrillIndicators`	Especifica se indicadores drill serão impressos com a tabela dinâmica.
`ShowDrillIndicators`	Especifica se os indicadores drill serão exibidos na tabela dinâmica.

continua

Tabela 13.2 Novas propriedades no Excel 2007 (*cont.*)

Propriedade	Descrição
`ShowTableStyleColumnHeaders`	Controla se o estilo Tabela 2 deve afetar os cabeçalhos de coluna.
`ShowTableStyleColumnStripes`	Controla se o estilo Tabela 2 deve exibir colunas em bandas.
`ShowTableStyleLastColumn`	Controla se o estilo Tabela 2 deve formatar a coluna final.
`ShowTableStyleRowHeaders`	Controla se o estilo Tabela 2 deve afetar os cabeçalhos de linha.
`ShowTableStyleRowStripes`	Controla se o estilo Tabela 2 deve exibir colunas em bandas.
`SortUsingCustomLists`	Controla se as listas personalizadas serão utilizadas para classificar itens de campos, tanto inicialmente como posteriormente à aplicação de uma classificação. Configurar essa propriedade como `False` pode otimizar o desempenho de campos com muitos itens e permitir que o uso de classificação baseada em lista personalizada seja evitado.
`TableStyle2`	Especifica o estilo de tabela dinâmica atualmente aplicado à tabela dinâmica. Observe que as versões anteriores do Excel ofereciam uma opção AutoFormatação fraca. As configurações desse recurso ficavam na propriedade `TableStyle`, então a Microsoft tinha de utilizar `TableStyle2` como o nome de propriedade para os novos estilos de tabela dinâmica. A propriedade poderia ter um valor como `PivotStyleLight17`.

Criando uma tabela dinâmica comum na interface Excel

Embora as tabelas dinâmicas sejam o recurso mais poderoso do Excel, a Microsoft estima que elas são utilizadas por apenas 7 por cento dos usuários do Excel. Com base em pesquisas no MrExcel.com, aproximadamente 42 por cento de usuários avançados do Excel usam tabelas dinâmicas. Como muitos de vocês nunca as utilizaram, apresento os passos para criar uma tabela dinâmica na interface com o usuário. Se você já for um profissional na criação de tabelas dinâmicas, pule para a próxima seção.

Digamos que você tenha 5 mil ou 500 mil linhas de dados, como mostrado na Figura 13.1. Você quer um resumo da receita por região e produto. As regiões são dispostas verticalmente, enquanto os produtos são dispostos horizontalmente.

	A	B	C	D	E	F	G	H
1	Região	Produto	Data	Cliente	Quantidade	Receita	Custo	Lucro
2	Oeste	D625	02/01/2007	Guarded Kettle Corporation	430	10937	6248	4689
3	Central	A292	02/01/2007	Mouthwatering Jewelry Company	400	8517	4564	3953
4	Oeste	B722	02/01/2007	Agile Glass Supply	940	23188	11703	11485
5	Central	E438	02/01/2007	Persuasive Kettle Inc.	190	5520	2958	2562
6	Leste	E438	02/01/2007	Safe Saddle Corporation	130	3933	2024	1909
7	Oeste	C409	02/01/2007	Agile Glass Supply	440	11304	5936	5368
8	Oeste	C409	02/01/2007	Guarded Kettle Corporation	770	20382	10387	9995
9	Central	E438	02/01/2007	Matchless Yardstick Inc.	570	17584	8875	8709
10	Leste	D625	02/01/2007	Unique Marble Company	380	10196	5521	4675
11	Central	D625	02/01/2007	Inventive Clipboard Corporation	690	18322	10026	8296
12	Oeste	E438	02/01/2007	Agile Glass Supply	580	16850	9031	7819
13	Oeste	A292	02/01/2007	Trouble-Free Eggbeater Inc.	560	11228	6276	4952
14	Central	C409	02/01/2007	Enhanced Toothpick Corporation	910	23143	12276	10867
15	Oeste	D625	02/01/2007	Trouble-Free Eggbeater Inc.	160	4561	2325	2236
16	Leste	E438	03/01/2007	Unique Marble Company	400	11740	6228	5512

Figura 13.1
Se precisar resumir rapidamente 500 mil linhas de dados transacionais, uma tabela dinâmica pode fazer isso em segundos. Seu objetivo é produzir um resumo de receita por região e produto.

Para construir a tabela dinâmica à direita dos dados, siga estes passos:

1. Selecione uma única célula nos dados de transação. Escolha o ícone Tabela Dinâmica da faixa Inserir. O Excel exibe a caixa de diálogo Criar Tabela Dinâmica.
2. Verifique se o Excel preencheu o intervalo da tabela com o endereço adequado. Contanto que seus dados não tenham linhas ou colunas completamente em branco, em geral, esse endereço estará correto.
3. Escolha criar a tabela dinâmica em uma planilha existente. Clique na caixa de Referência Local e escolha célula J1, como mostrado na Figura 13.2.

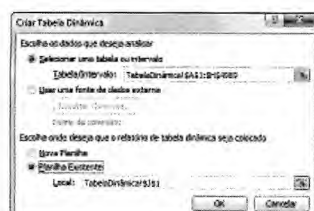
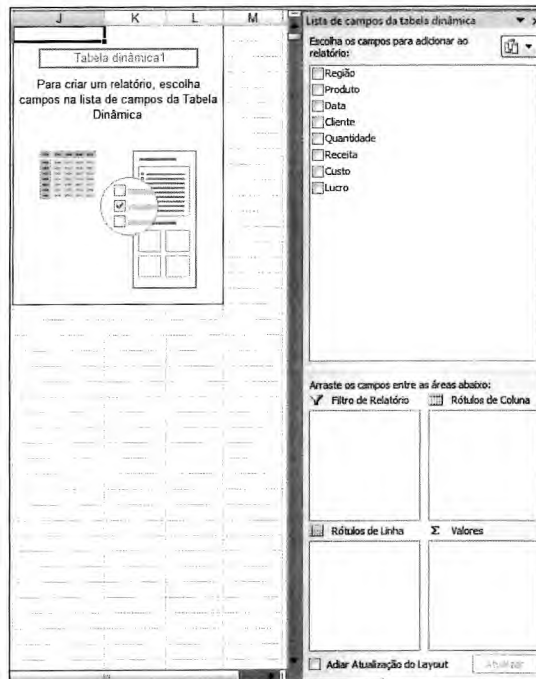
Figura 13.2
Verifique se o Excel selecionou os dados corretamente e especifique uma localização para a tabela dinâmica.

4. Clique em OK para criar uma tabela dinâmica em branco. As instruções nessa tabela em branco solicitam que você escolha campos da Lista de Campos da Tabela Dinâmica. Essa lista aparece no lado direito da tela. Uma lista dos campos disponíveis está na parte superior do painel de tarefas. Quatro áreas, chamadas Filtro de Relatório, Rótulos de Coluna, Rótulos de Linha e Valores, aparecem na parte inferior do painel de tarefas (veja Figura 13.3).

Figura 13.3
O Excel apresenta uma lista de campos disponíveis e quatro áreas na Lista de Campos da Tabela Dinâmica.

5. Clique nos campos Região e Receita na seção superior da Lista de Campos da Tabela Dinâmica. Como o campo Região contém dados de texto, ele é automaticamente movido para a área Rótulos de Linha. Como o campo Receita contém dados numéricos, é automaticamente movido para a área Valores.

6. Clique no campo Produto na seção principal da Lista de Campos da Tabela Dinâmica e arraste-o para a área Rótulos de Coluna na parte inferior da Lista de Campos da Tabela Dinâmica. Isso adiciona uma lista de produtos que se estende pela primeira linha da tabela dinâmica.

Como mostrado na Figura 13.4, o Excel criou um resumo conciso de seus dados na tabela dinâmica.

Depois de criar uma tabela dinâmica na planilha, você pode alterar facilmente os dados resumidos no relatório arrastando os campos dentro das áreas da Lista de Campos da Tabela Dinâmica. Na Figura 13.5, o cliente foi adicionado à seção Rótulos de Linha da tabela dinâmica existente.

Soma de Receita	Rótulos de Coluna					
Rótulos de Linha	A292	B722	C409	D625	E438	Total geral
Central	4043186	4165988	4309384	4619765	5854458	22992781
Leste	4153030	4301310	4983303	5167376	5726406	24331425
Oeste	4141562	4215763	4809076	4782819	5831102	23780322
Total geral	12337778	12683061	14101763	14569960	17411966	71104528

Figura 13.4
Apenas seis cliques foram necessários para criar esse resumo.

Entendendo os novos recursos das tabelas dinâmicas do Excel 2007

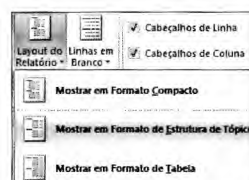
Por padrão, todas as novas tabelas dinâmicas são criadas em um novo layout chamado Formulário Compacto. Nesse layout, vários campos Linha aparecem em uma única coluna à esquerda da tabela dinâmica. O Excel também coloca os subtotais acima das linhas de detalhes.

Soma de Receita	Rótulos de Coluna			
Rótulos de Linha	Central	Leste	Oeste	Total geral
A292	4043186	4153030	4141562	12337778
Agile Glass Supply			628204	628204
Enhanced Toothpick Corporation	293017			293017
Excellent Glass Traders		447771		447771
Functional Shingle Corporation			504818	504818
Guarded Kettle Corporation			1450110	1450110
Innovative Oven Corporation			452320	452320
Inventive Clipboard Corporation	410968			410968
Magnificent Patio Traders		395186		395186
Matchless Yardstick Inc.	476223			476223
Mouthwatering Jewelry Company	374000			374000
Mouthwatering Tripod Corporation		337100		337100
Persuasive Kettle Inc.	1565368			1565368
Persuasive Yardstick Corporation			268394	268394
Remarkable Umbrella Company	362851			362851
Safe Saddle Corporation		646559		646559
Tremendous Bobsled Corporation	560759			560759
Tremendous Flagpole Traders			446799	446799
Trouble-Free Eggbeater Inc.			390917	390917
Unique Marble Company		1600347		1600347
Unique Saddle Inc.		408114		408114
Vibrant Tripod Corporation		317953		317953
B722	4165988	4301310	4215763	12683061
Agile Glass Supply			652845	652845
Enhanced Toothpick Corporation	403764			403764
Excellent Glass Traders		386804		386804
Functional Shingle Corporation			289670	289670
Guarded Kettle Corporation			1404742	1404742
Innovative Oven Corporation			364200	364200

Figura 13.5
O nome *tabela dinâmica* vem da capacidade de arrastar os campos nas áreas e fazê-los recalcular. Com dois cliques, você pode mover a Região pela parte superior, mover Produto para a parte inferior e adicionar um resumo por Cliente.

Embora essas alterações possam resultar em uma tabela dinâmica mais ativa, a maioria das tabelas dinâmicas neste capítulo será convertida em valores para produzir um resumo estático dos dados. Nesses casos, você quer reverter para o layout Estrutura de Tópicos, que era o padrão no Excel 2003. Os próximos passos na interface com o usuário superam essas novas escolhas padrão:

1. Na guia Design, escolha Layout do Relatório, Mostrar em Formato de Estrutura de Tópicos, como mostrado na Figura 13.6.
2. Na faixa Design, escolha Subtotais, Mostrar Todos os Subtotais no Final do Grupo.
3. Na faixa Opções, escolha Tabela Dinâmica no lado esquerdo e o ícone Opções. Na guia Layout & Formato da caixa de diálogo Opções da Tabela Dinâmica, digite um zero ao lado de Para células vazias, mostrar.

Figura 13.6
Se planeja reutilizar o resultado da tabela dinâmica, você deve mudar de Formato Compacto para Estrutura de Tópicos, a fim de que cada campo Linha tenha sua própria coluna.

Criando uma tabela dinâmica no Excel VBA

Neste capítulo, não quis dizer que utilizamos o VBA para construir tabelas dinâmicas para nossos clientes. Na verdade, a finalidade deste capítulo é lembrá-lo de que as tabelas dinâmicas podem ser utilizadas como um meio para alcançar um propósito; você pode utilizá-las para extrair um resumo de dados e, então, utilizar esse resumo para outra tarefa.

> **NOTA**
> O download das listagens de código deste capítulo está disponível em www.prenhall.com/jelen_br ou www.mrexcel.com/getcode2007.html.
> Embora a interface com o usuário do Excel tenha novos nomes para as várias seções de uma tabela dinâmica, o código VBA continua a se referir aos nomes antigos. A Microsoft teve de optar por essa escolha; caso contrário, milhões de linhas de código parariam de funcionar no Excel 2007 quando se referissem a um Campo de Página em vez de a um Campo de Filtro. Embora as quatro seções de uma tabela dinâmica na interface com o usuário do Excel sejam Filtro de Relatório, Rótulos de Coluna, Rótulos de Linha e Valores, o VBA continua a utilizar os antigos termos de Campos de Página, Coluna, Linha e Dados.

Definindo o cache dinâmico

No Excel 2000 e versões posteriores, você primeiro cria um objeto cache dinâmico para descrever a área de entrada dos dados:

```
Dim WSD As Worksheet
Dim PTCache As PivotCache
Dim PT As PivotTable
Dim PRange As Range
Dim FinalRow As Long
Dim FinalCol As Long
Set WSD = Worksheets("TabelaDinâmica")

'Exclui todas as tabelas dinâmicas anteriores
For Each PT In WSD.PivotTables
    PT.TableRange2.Clear
Next PT

'Define a área de entrada e configure um cache Pivot
FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
FinalCol = WSD.Cells(1, Columns.Count).End(xlToLeft).Column
Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase,SourceData:=PRange)
```

Criando e configurando a tabela dinâmica

Depois de definir o cache dinâmico, utilize o método `CreatePivotTable` para criar uma tabela dinâmica em branco, baseada no cache dinâmico definido:

```
Set PT = PTCache.CreatePivotTable(TableDestination:=WSD.Cells(2, FinalCol + 2),TableName:="Tabela dinâmica1")
```

No método `CreatePivotTable`, você especifica a localização de saída e, opcionalmente, dá um nome à tabela. Depois de executar essa linha de código, a tabela dinâmica adquire uma aparência estranha, como mostrado na Figura 13.7. Agora você utiliza código para soltar campos na tabela.

Figura 13.7
Quando você utiliza o método `CreatePivotTable`, o Excel apresenta uma, não muito útil, tabela dinâmica em branco de quatro células.

Se escolher a configuração Adiar Atualização de Layout na interface com o usuário para construir a tabela dinâmica, o Excel não recalculará a tabela dinâmica depois de você soltar todos os campos sobre a tabela. Por padrão, no VBA, o Excel calcula a tabela dinâmica enquanto você executa cada passo na criação da tabela. Isso poderia exigir que a tabela dinâmica fosse executada várias vezes antes que você chegasse ao resultado final. Para acelerar a execução do código, você pode desativar temporariamente o cálculo da tabela dinâmica utilizando a propriedade `ManualUpdate`:

```
PT.ManualUpdate = True
```

Você agora pode executar os passos necessários para organizar a tabela dinâmica. No método `.AddFields`, podemos especificar um ou mais campos que devem estar na área de linha, coluna ou filtro da tabela dinâmica.

O parâmetro `RowFields` permite definir campos que aparecem na área Rótulos de Linha da Lista de Campo de Tabela Dinâmica. O parâmetro `ColumnFields` corresponde à área de Rótulos de Coluna. O parâmetro `PageFields` corresponde à área de Filtro de Relatório.

A linha de código seguinte preenche uma tabela dinâmica com dois campos na área de linha e um na área de coluna:

```
'Configurando campos de linha & coluna
PT.AddFields RowFields:=Array("Região", "Cliente"),ColumnFields:="Produto"
```

Para adicionar um campo como Receita à área de valores da tabela, mude a propriedade `Orientation` do campo para `xlDataField`.

Obtendo uma soma em vez de uma contagem

O Excel é inteligente. Quando você cria um relatório com receita, ele supõe que você quer somar a receita. Mas há um problema. Suponha que uma das células de receita esteja em branco. Quando você cria a tabela dinâmica, mesmo que 9,9 por cento de campos sejam numérico, o Excel supõe que você tem dados alfanuméricos e se oferece para contar esse campo. Isso é irritante. Parece uma anomalia que, de um lado, espera-se que você certifique-se de que 100 por cento das células tenham dados numéricos; de outro, porém, os resultados da tabela dinâmica são freqüentemente preenchidos com células em branco não-numéricas.

Ao fazer a tabela dinâmica na interface do Excel, você deve tomar cuidado na seção □ Valores e notar que o campo exibe Contagem de Receita em vez de Soma de Receita. Nesse ponto, o curso certo de ação é voltar e corrigir os dados, mas o que as pessoas normalmente fazem é dar um clique duplo no botão Contagem de Receita, mudando para Soma de Receita.

No VBA, você deve sempre definir explicitamente que está criando uma soma de receita configurando explicitamente a propriedade `Function` como `xlSum`:

```
'Configura os campos de dados
With PT.PivotFields("Receita")
    .Orientation = xlDataField
    .Function = xlSum
    .Position = 1
End With
```

Nesse ponto, você forneceu todas as configurações necessárias para o VBA gerar corretamente a tabela dinâmica. Se configurar `ManualUpdate` como `False`, o Excel calcula e desenha a tabela dinâmica. Logo depois você pode configurar isso de novo como `True`:

```
'Calcula a tabela dinâmica
PT.ManualUpdate = False
PT.ManualUpdate = True
```

Sua tabela dinâmica herda as configurações de estilo de tabela selecionadas como o padrão em qualquer computador que venha executar o código. Se quiser controlar o formato final, você pode escolher explicitamente um estilo de tabela. O código seguinte aplica linhas em banda e um estilo médio de tabela:

```
'Formata a tabela dinâmica
PT.ShowTableStyleRowStripes = True
PT.TableStyle2 = "PivotStyleMedium10"
```

Nesse ponto, a tabela dinâmica está completa e é mostrada na Figura 13.8.

Figura 13.8
Menos de 50 linhas de código criam essa tabela dinâmica em menos de um segundo.

Soma de Receita		Produto
Região	Cliente	A292
⊟Central	Enhanced Toothpick Corporation	293017
	Inventive Clipboard Corporation	410968
	Matchless Yardstick Inc.	476223
	Mouthwatering Jewelry Company	374000
	Persuasive Kettle Inc.	1565368
	Remarkable Umbrella Company	362851
	Tremendous Bobsled Corporation	560759
Central Total		4043186
⊟Leste	Excellent Glass Traders	447771

A Listagem 13.1 mostra o código completo utilizado para gerar a tabela dinâmica.

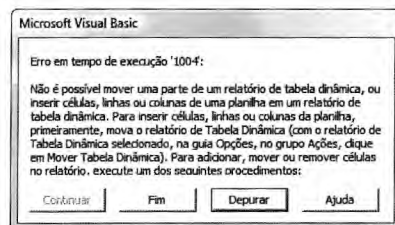
Listagem 13.1 Código para gerar uma tabela dinâmica

```
Sub CreatePivot()
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long
    Set WSD = Worksheets("TabelaDinâmica")

    ' Exclui todas as tabelas dinâmicas anteriores
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT

    ' Define a área de entrada e configura um cache dinâmico
```

```
    FinalRow = WSD.Cells(Application.Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Application.Columns.Count).End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, SourceData:=PRange.Address)

    ' Cria a tabela dinâmica a partir do cache dinâmico
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD.Cells(2, FinalCol + 2), TableName:="Tabela _
          dinâmica1")

    ' Desativa a atualização ao criar a tabela
    PT.ManualUpdate = True

    ' Configura os campos Linha & Coluna
    PT.AddFields RowFields:=Array("Região", "Cliente"),ColumnFields:="Produto"

    ' Configura os campos de dados
    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
    End With

    ' Calcula a tabela dinâmica
    PT.ManualUpdate = False
    PT.ManualUpdate = True

    'Formata a tabela dinâmica
    PT.ShowTableStyleRowStripes = True
    PT.TableStyle2 = "PivotStyleMedium10"

    WSD.Activate
    Range("J2").Select
End Sub
```

Aprendendo por que não é possível mover nem alterar parte de um relatório dinâmico

Embora as tabelas dinâmicas sejam incríveis, elas têm limitações irritantes. Não é possível mover nem alterar apenas parte de uma tabela dinâmica. Por exemplo, tente executar uma macro que excluiria a coluna Q, que contém a coluna Total Geral da tabela. A macro chega a uma interrupção ruidosa com um erro 1004, como mostrado na Figura 13.9. Para driblar essa limitação, você pode mudar o resumo de uma tabela dinâmica para valores apenas.

Figura 13.9
Você não consegue excluir parte de uma tabela dinâmica.

Determinando o tamanho de uma tabela dinâmica pronta

É difícil saber de antemão o tamanho de uma tabela dinâmica. Se executar um relatório de dados transacionais um dia, você pode ou não obter as vendas da região Oeste, por exemplo. Isso poderia fazer com que a tabela tivesse seis ou sete colunas de largura. Portanto, você deve utilizar a propriedade `TableRange2` especial para referir-se à tabela dinâmica resultante inteira.

Por causa das limitações das tabelas dinâmicas, em geral, deve-se copiar os resultados de uma tabela dinâmica para um novo local na planilha e depois excluir a tabela dinâmica original. O código em `CreateSummaryReportUsingPivot()` cria uma pequena tabela dinâmica. Observe que é possível configurar as propriedades `ColumnGrand` e `RowGrand` da tabela como `False` para impedir que os totais sejam adicionados à tabela.

`PT.TableRange2` inclui a tabela dinâmica inteira. Na Figura 13.10, TableRange2 inclui a linha extra no começo da tabela junto com o botão Soma da Receita. Para eliminar essa linha, o código copia `PT.TableRange2`, mas desloca essa seleção uma linha utilizando `.Offset(1, 0)`. Dependendo da natureza da tabela dinâmica, talvez você precise usar um deslocamento de duas ou mais linhas para eliminar as informações irrelevantes da parte superior da tabela dinâmica.

O código copia `PT.TableRange2` e utiliza `PasteSpecial` em uma célula de cinco linhas abaixo da tabela dinâmica atual. Nesse ponto do código, a planilha é semelhante à mostrada na Figura 13.10. A tabela em J 2 é uma tabela dinâmica ativa e a tabela em J12 tem apenas os resultados copiados.

	J	K	L	M
	Soma de Receita	Região		
	Produto	Central	Leste	Oeste
	A292	4043186	4153030	4141562
	B722	4165988	4301310	4215763
	C409	4309384	4983303	4809076
	D625	4619765	5167376	4782819
	E438	5854458	5726406	5831102

Produto	Central	Leste	Oeste
A292	4043186	4153030	4141562
B722	4165988	4301310	4215763
C409	4309384	4983303	4809076
D625	4619765	5167376	4782819
E438	5854458	5726406	5831102

Figura 13.10
Essa figura mostra um resultado intermediário da macro. Apenas o resumo em J12:M17 permanecerá depois que a macro terminar.

Você pode então eliminar totalmente a tabela dinâmica aplicando o método `Clear` à tabela inteira. Se seu código continuar fazendo formatação adicional, você deve remover o cache dinâmico da memória configurando `PTCache` igual a `Nothing`.

O código na Listagem 13.2 utiliza uma tabela dinâmica para produzir um resumo dos dados subjacentes. No fim do código, a tabela dinâmica será copiada para valores estáticos e a tabela dinâmica será limpada.

Listagem 13.2 Código para produzir um resumo estático de uma tabela dinâmica

```
Sub CreateSummaryReportUsingPivot()
    ' Utiliza uma tabela dinâmica para criar um relatório de resumo estático
    ' com Produto linhas abaixo e as regiões pelas colunas
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long
    Set WSD = Worksheets("TabelaDinâmica")

    ' Exclui todas as tabelas dinâmicas anteriores
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("J1:Z1").EntireColumn.Clear

    ' Define a área de entrada e configura um cache dinâmico
    FinalRow = WSD.Cells(Application.Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Application.Columns.Count).End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, SourceData:=PRange.Address)

    ' Cria a tabela dinâmica a partir do cache dinâmico
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD.Cells(2, FinalCol + 2), TableName:="Tabela _
        dinâmica1")

    ' Desativa a atualização ao criar a tabela
    PT.ManualUpdate = True

    ' Configura os campos de linha
    PT.AddFields RowFields:="Produto", ColumnFields:="Região"

    ' Configura os campos de dados
    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
    End With

    With PT
        .ColumnGrand = False
        .RowGrand = False
        .NullString = "0"
    End With

    ' Calcula a tabela dinâmica
    PT.ManualUpdate = False
    PT.ManualUpdate = True
```

```
    ' PT.TableRange2 contém os resultados. Move esses para J12
    ' como valores apenas e não como uma tabela dinâmica real.
    PT.TableRange2.Offset(1, 0).Copy
    WSD.Cells(5 + PT.TableRange2.Rows.Count, FinalCol + 2).PasteSpecial xlPasteValues

    ' Nesse ponto, a planilha é semelhante à Figura 13.10
    ' Redefine

    ' Exclui a tabela dinâmica original & cache dinâmico
    PT.TableRange2.Clear
    Set PTCache = Nothing

    WSD.Activate
    Range("J12").Select
End Sub
```

O código na Listagem 13.2 cria a tabela dinâmica. Depois o código copia os resultados como valores e os cola, também como valores, em J12:M13. A Figura 13.10 (mostrada anteriormente) exibe um resultado intermediário antes da tabela dinâmica original ser limpada.

Até agora, este capítulo o orientou pelas etapas da criação dos mais simples relatórios de tabela dinâmica. As tabelas dinâmicas oferecem muito mais flexibilidade. As próximas seções apresentam exemplos de relatórios mais complexos.

Criando um relatório para mostrar receita por produto

Um típico relatório poderia fornecer uma lista de regiões por produto com receita anual. Esse relatório poderia ser fornecido aos gerentes de linhas de produtos para mostrar-lhes quais clientes estão comprando seus produtos. Nesse exemplo, você quer apresentar os clientes em ordem decrescente por receita e o campo de anos pelas colunas. A Figura 13.11 mostra um relatório de exemplo.

Figura 13.11
Uma típica solicitação é considerar dados transacionais e fazer um resumo por produto para gerentes de linhas de produtos. Você pode utilizar uma tabela dinâmica para obter 90 por cento desse relatório e depois concluí-lo com uma pequena formatação.

	A	B	C	D	E
1	Receita por Cliente e Ano				
2					
3	Produto	Cliente	2007	2008	Total geral
4	A292	Unique Marble Company	730K	871K	1.600K
5	A292	Persuasive Kettle Inc.	868K	697K	1.565K
6	A292	Guarded Kettle Corporation	711K	739K	1.450K
7	A292	Safe Saddle Corporation	184K	462K	647K
8	A292	Agile Glass Supply	354K	275K	628K
9	A292	Tremendous Bobsled Corporation	305K	256K	561K
10	A292	Functional Shingle Corporation	0K	505K	505K
11	A292	Matchless Yardstick Inc.	264K	212K	476K
12	A292	Innovative Oven Corporation	263K	189K	452K
13	A292	Excellent Glass Traders	177K	271K	448K
14	A292	Tremendous Flagpole Traders	169K	278K	447K
15	A292	Inventive Clipboard Corporation	208K	203K	411K
16	A292	Unique Saddle Inc.	258K	150K	408K
17	A292	Magnificent Patio Traders	245K	150K	395K
18	A292	Trouble-Free Eggbeater Inc.	162K	229K	391K
19	A292	Mouthwatering Jewelry Company	211K	163K	374K
20	A292	Remarkable Umbrella Company	199K	163K	363K
21	A292	Mouthwatering Tripod Corporation	138K	199K	337K
22	A292	Vibrant Tripod Corporation	124K	194K	318K
23	A292	Enhanced Toothpick Corporation	148K	145K	293K
24	A292	Persuasive Yardstick Corporation	92K	176K	268K
25	A292 Total		5.811K	6.526K	12.338K
26	B722	Unique Marble Company	755K	827K	1.582K
27	B722	Guarded Kettle Corporation	843K	562K	1.405K
28	B722	Persuasive Kettle Inc.	728K	657K	1.385K

A chave para produzir esses dados rapidamente é utilizar uma tabela dinâmica. Embora elas sejam ótimas para resumir dados, em geral, você terá de executar alguns passos adicionais para alcançar o resultado desejado.

Para criar esse relatório, inicie com uma tabela dinâmica que tenha Produto e Cliente como campos Linha, Data agrupados por ano como campo Coluna e Soma de Receita como campo Dados. A Figura 13.12 mostra a tabela dinâmica padrão criada com essas configurações.

Eis apenas algumas das dificuldades que a maioria das tabelas dinâmicas apresenta por padrão:

- A visualização Estrutura de Tópicos é horrível. Na Figura 13.12, o valor A292 aparece na coluna Produto apenas uma vez e é seguido por 20 células em branco. Esse é o pior recurso das tabelas dinâmicas e não há absolutamente nada para corrigi-lo. Embora as pessoas possam entender que esta seção inteira se destina às vendas A292, é muito confuso se sua seção A292 recorre para uma segunda ou terceira página. A página 2 inicia sem nenhuma indicação de que o relatório é sobre as vendas A292. Se pretende redirecionar (*repurpose*) os dados, você precisa que o valor das vendas A292 estejam em todas as linhas.

Figura 13.12
Utilize o poder da tabela dinâmica para resumir os dados, mas depois use seu bom senso para formatar o relatório.

	J	K	L	M	N
1	Soma de Receita		Data		
2	Produto	Cliente	2007	2008	Total geral
3	⊟A292	Agile Glass Supply	353678	274526	628204
4		Enhanced Toothpick Corporation	148419	144598	293017
5		Excellent Glass Traders	176704	271067	447771
6		Functional Shingle Corporation		504818	504818
7		Guarded Kettle Corporation	710732	739378	1450110
8		Innovative Oven Corporation	262822	189498	452320
9		Inventive Clipboard Corporation	207939	203029	410968
10		Magnificent Patio Traders	245236	149950	395186
11		Matchless Yardstick Inc.	263819	212404	476223
12		Mouthwatering Jewelry Company	211285	162715	374000
13		Mouthwatering Tripod Corporation	138413	198687	337100
14		Persuasive Kettle Inc.	868363	697005	1565368
15		Persuasive Yardstick Corporation	92341	176053	268394
16		Remarkable Umbrella Company	199373	163478	362851
17		Safe Saddle Corporation	184144	462415	646559
18		Tremendous Bobsled Corporation	304831	255928	560759
19		Tremendous Flagpole Traders	169043	277756	446799
20		Trouble-Free Eggbeater Inc.	162233	228684	390917
21		Unique Marble Company	729836	870511	1600347
22		Unique Saddle Inc.	258034	150080	408114
23		Vibrant Tripod Corporation	124242	193711	317953
24	A292 Total		5811487	6526291	12337778
25	⊟B722	Agile Glass Supply	436866	215979	652845
26		Enhanced Toothpick Corporation	189918	213846	403764

- O relatório contém células em branco em vez de zeros. Na Figura 13.12, a Functional Shingle não fez nenhuma venda em 2007. O Excel produz uma tabela dinâmica em que a célula L6 está em branco em vez de ter o valor zero. Essa é simplesmente uma forma ruim. Especialistas em Excel contam que podem percorrer o intervalo utilizando as teclas End e as teclas de seta. As células em branco destroem essa capacidade.
- O título é enfadonho. A maioria das pessoas concorda que Soma de Receita é um título irritante.
- Algumas legendas são irrelevantes. A data flutuante na célula L1 da Figura 13.12 realmente não pertence a um relatório.
- Raramente a ordem de classificação alfabética padrão é útil. Os gerentes de linha de produtos preferirão que os principais clientes sejam apresentados no início da lista. Seria útil classificar o relatório em ordem decrescente por receita.
- Dependendo do estilo-padrão da tabela dinâmica do computador, as bordas podem ter uma aparência desagradável. O Excel tem uma grande quantidade de bordas que, de fato, deixam o relatório muito feio.
- O formato de número padrão é Geral. Seria melhor definir esses dados adotando as vírgulas como separadores de milhares ou até mesmo dados em milhares ou milhões.
- As tabelas dinâmicas não oferecem uma lógica óbvia de quebra de página. Se quiser gerar relatórios para gerentes de cada linha de negócios, você teria de se aprofundar na última configuração da guia Voltar da caixa de diálogo Configurações de Campo.
- Por causa do problema da quebra de página, talvez você ache mais fácil se desfazer das linhas de subtotal da tabela dinâmica e fazer o método `Subtotal` adicionar linhas de subtotal com quebras de página. Você precisa de um modo de desativar as linhas de subtotal da tabela dinâmica oferecidas por Produto na Figura 13.12. Essas linhas aparecem automaticamente sempre que há dois ou mais campos de linha. Se houvesse quatro campos Linha, você poderia querer desativar os subtotais automáticos dos três campos Linha mais externos.

Mesmo com todos esses problemas nas tabelas dinâmicas padrão, elas ainda são muito úteis. Você pode superar todas as reclamações, utilizando configurações especiais dentro da tabela dinâmica ou inserindo algumas linhas de código depois que a tabela dinâmica é criada e copiada para um conjunto de dados comum.

Eliminando células em branco na área de valores

As pessoas começaram a reclamar das células em branco logo que as tabelas dinâmicas foram lançadas. Qualquer pessoa que utilize o Excel 97 ou versão posterior pode facilmente substituir células em branco por zeros. Na interface com o usuário, você pode localizar a configuração na guia Layout & Formato da caixa de diálogo Opções de Tabela Dinâmica. Escolha a opção Para Células Vazias, Mostrar e digite `0` na caixa.

A operação equivalente no VBA é configurar a propriedade `NullString` para a tabela dinâmica com `"0"`.

NOTA Embora o *código adequado seja configurar* esse valor como um zero de texto, o Excel realmente coloca um zero real nas células vazias.

Assegurando o uso do layout de tabela

Nas versões do Excel anteriores a 2007, múltiplos campos Linha apareciam em múltiplas colunas. Três layouts estão disponíveis no Excel 2007. O layout Compacto comprime todos os campos Linha em uma única coluna.

Para impedir esse resultado e assegurar que sua tabela dinâmica esteja no layout clássico, utilize este código:

```
PT.RowAxisLayout xlTabularRow
```

Controlando a ordem de classificação com AutoClassificação

A interface com o usuário do Excel oferece uma opção AutoClassificação que permite mostrar mercados em ordem decrescente com base na receita. O código equivalente em VBA para classificar o campo de produto por receita descendente utiliza o método `AutoSort`:

```
PT.PivotFields("Cliente").AutoSort Order:=xlDescending,Field:="Soma de Receita"
```

Alterando o formato de número padrão

Para alterar o formato de número na interface com o usuário, escolha um campo de receita e, na faixa Opções, escolha Campo Ativo, Configurações de Campo, Formato de Número. Então escolha um formato de número apropriado.

Quando você tiver números grandes, exibir o separador de milhares auxilia a leitura do relatório. Para configurar esse formato no código VBA, utilize o seguinte:

```
PT.PivotFields("Soma de Receita").NumberFormat = "#,##0"
```

Algumas empresas têm clientes que em geral compram milhares ou milhões dólares em mercadorias. Você pode exibir números em milhares utilizando uma única vírgula depois do formato de número. Naturalmente, você precisa incluir uma abreviação `K` para indicar que os números estão em milhares:

```
PT.PivotFields("Soma de Receita").NumberFormat = "#,##0,K"
```

O local personalizado dita a abreviação de milhares. Se estiver trabalhando para uma empresa de informática relativamente jovem, em que todos usam `K` como o separador de milhares, você tem sorte porque a Microsoft facilita o uso dessa abreviação. Mas, se trabalha em uma empresa de sabão com mais de 100 anos, na qual você utiliza `M` para milhares e `MM` para milhões, terá de superar mais alguns obstáculos. Você precisa prefixar o caractere `M` com uma barra invertida para que isso funcione:

```
PT.PivotFields("Soma de Receita").NumberFormat = "#,##0,\M"
```

Alternativamente, você pode colocar o caractere `M` entre aspas duplas. Para colocar uma string citada no VBA entre aspas duplas, coloque duas aspas seqüenciais. Para configurar um formato em décimo de milhões que utiliza o formato `#,##0.0,,"MM`, utilize essa linha de código:

```
PT.PivotFields("Soma de Receita").NumberFormat = "#,##0.0,,""M"""
```

No caso, isso é difícil de ler. O formato para o código é aspa, libra, vírgula, libra, libra, zero, ponto, zero, vírgula, vírgula, aspa, aspa, M, aspa, aspa, aspa. As três aspas no fim são corretas. Você utiliza duas aspas para simular uma aspa na caixa personalizada de formato de número e uma aspa final para fechar a string em VBA.

Removendo subtotais de vários campos Linha

Assim que você tiver mais de um campo Linha, o Excel adiciona automaticamente subtotais a todos os campos, exceto ao mais interno de Linha. Mas talvez você queira remover subtotais por várias razões. Embora realizar essa tarefa de maneira manual possa ser relativamente simples, o código do VBA para suprimir subtotais é surpreendentemente complexo.

Deve-se configurar a propriedade `Subtotals` igual a um array de 12 valores `False`. Consulte a ajuda do VBA para obter todos os detalhes, mas isso é algo assim: o primeiro valor `False` desativa subtotais automáticos, o segundo `False` desativa o subtotal `Sum`, o terceiro `False` desativa o subtotal `Count` e assim por diante. É interessante que você tenha de desativar todos 12 possíveis subtotais, mesmo que o Excel exiba apenas um. Essa linha de código suprime o subtotal `Produto`:

```
PT.PivotFields("Produto").Subtotals = Array(False, False, False, False,False, False, False, False, False, False,
False, False)
```

Uma técnica diferente é ativar o primeiro subtotal. Esse método desativa automaticamente os outros 11 subtotais. A partir daí, é possível desativar o primeiro subtotal para certificar-se de que todos os subtotais serão eliminados:

```
PT.PivotFields("Produto").Subtotals(1) = True
PT.PivotFields("Produto").Subtotals(1) = False
```

Removendo o Total Geral de linhas

Como você vai usar o código do VBA para adicionar subtotais automáticos, elimine a linha Total Geral. Se desativar o Total Geral das Linhas, você exclui a coluna chamada Total Geral. Portanto, para eliminar a linha Total Geral, desmarque Total Geral das Colunas. Isso é tratado no código com a linha a seguir:

```
PT.ColumnGrand = False
```

Tratando dificuldades adicionais ao criar seu relatório final

Você chegou ao fim dos ajustes que poderia fazer à tabela dinâmica. Para terminar o relatório final, é preciso fazer os ajustes restantes depois de converter a tabela dinâmica em dados comuns.

A Figura 13.13 mostra a tabela dinâmica com todos os ajustes descritos nas seções anteriores e com `PT.TableRange2` selecionado.

Figura 13.13
Fazer 90 por cento do relatório levou menos de um segundo e menos 30 de linhas de código. Para resolver os últimos cinco problemas irritantes, você precisa transformar esses dados de uma tabela dinâmica em dados regulares.

Soma de Receita		Data		
Produto	Cliente	2007	2008	Total geral
A292	Unique Marble Company	730K	871K	1.600K
	Guarded Kettle Corporation	711K	739K	1.450K
	Persuasive Kettle Inc.	868K	697K	1.565K
	Safe Saddle Corporation	184K	462K	647K
	Agile Glass Supply	354K	275K	628K
	Tremendous Bobsled Corporation	305K	256K	561K
	Functional Shingle Corporation		505K	505K
	Matchless Yardstick Inc.	264K	212K	476K
	Innovative Oven Corporation	263K	189K	452K
	Excellent Glass Traders	177K	271K	448K
	Tremendous Flagpole Traders	169K	278K	447K
	Inventive Clipboard Corporation	208K	203K	411K
	Unique Saddle Inc.	258K	150K	408K
	Magnificent Patio Traders	245K	150K	395K
	Trouble-Free Eggbeater Inc.	162K	229K	391K
	Mouthwatering Jewelry Company	211K	163K	374K
	Remarkable Umbrella Company	199K	163K	363K
	Mouthwatering Tripod Corporation	138K	199K	337K
	Vibrant Tripod Corporation	124K	194K	318K
	Enhanced Toothpick Corporation	148K	145K	293K
	Persuasive Yardstick Corporation	92K	176K	268K
B722	Unique Marble Company	755K	827K	1.582K
	Guarded Kettle Corporation	843K	562K	1.405K

Criando uma nova pasta de trabalho para armazenar o relatório

Digamos que você queira criar o relatório em uma nova pasta de trabalho para que ele possa ser facilmente enviado por e-mail para os gerentes de produtos. É relativamente fácil fazer isso. Para tornar o código mais portável, atribua as variáveis de objeto à pasta de trabalho original, à nova pasta de trabalho e à primeira planilha na nova pasta de trabalho. No início do procedimento, adicione essas instruções:

```
Dim WSR As Worksheet
Dim WBO As Workbook
Dim WBN As Workbook
Set WBO = ActiveWorkbook
Set WSD = Worksheets("TabelaDinâmica")
```

Depois de a tabela dinâmica ser criada com sucesso, faça uma pasta de trabalho de Relatório em branco com este código:

```
'Cria uma nova pasta de trabalho em branco com uma planilha
Set WBN = Workbooks.Add(xlWorksheet)
Set WSR = WBN.Worksheets(1)
WSR.Name = "Relatório"
'Configura o título do relatório
With WSR.Range("A1")
    .Value = "Receita por cliente e ano"
    .Font.Size = 14
End With
```

Criando um resumo sobre uma planilha de relatório em branco

Imagine que você tivesse enviado a tabela dinâmica ilustrada na Figura 13.13 e seu gerente odiasse as bordas, o título e a palavra *Data* na célula L2. Você resolveria esses problemas excluindo a(s) primeira(s) linha(s) de `PT.TableRange2` do método `.Copy` e utilizando `PasteSpecial(xlPasteValuesAndNumberFormats)` para copiar os dados para a planilha de relatório.

DICA

No Excel 2000 e versões anteriores, `xlPasteValuesAndNumberFormats` não estava disponível. Precisávamos utilizar o comando Colar Especial duas vezes: uma vez como `xlPasteValues` e a outra como `xlPasteFormats`.

No exemplo atual, a propriedade `.TableRange2` inclui apenas uma linha a ser eliminada, a Linha 2, como mostrado na Figura 13.13. Se tivesse uma tabela dinâmica mais complexa com vários campos Coluna ou um ou mais campos de página, você teria de eliminar apenas a primeira linha do relatório. Isso ajuda a executar a macro até esse ponto, a examinar o resultado e a descobrir quantas linhas precisam ser excluídas. Não é possível copiar efetivamente essas linhas para o relatório utilizando a propriedade `Offset`. Copie a propriedade `TableRange2`, desloque uma única linha. Os puristas notam que esse código copia uma linha extra em branco embaixo da tabela dinâmica, mas isso não importa realmente, porque essa é uma linha em branco. Depois de copiar, você pode apagar a tabela dinâmica original e destruir o cache dinâmico:

```
'Copia os dados de tabela dinâmica para a linha 3 da planilha Relatório
'Usa Offset para eliminar a linha de título da tabela dinâmica
PT.TableRange2.Offset(1, 0).Copy
WSR. Range("A3").PasteSpecial Paste:=xlPasteValuesAndNumberFormats
PT.TableRange2.Clear
Set PTCache = Nothing
```

Observe que você utiliza a opção Colar Especial para colar apenas valores e formatos de número. Isso elimina tanto as bordas como a natureza dinâmica da tabela. Talvez você fique tentado a usar a opção Sem Bordas em Colar, mas isso mantém os dados em uma tabela dinâmica, e você não conseguirá inserir novas linhas no meio dos dados.

Preenchendo a visualização Estrutura de Tópicos

O relatório está quase completo. Você está a quase um comando Dados, Subtotais, de distância de obter tudo de que precisa. Antes de utilizar o comando Subtotais, porém, é preciso preencher todas as células em branco no modo de visualização Estrutura de Tópicos da coluna A.

Corrigir a visualização Estrutura de Tópicos requer apenas alguns passos obscuros. Eis os passos na interface com o usuário:

1. Selecione todas as células na coluna A que compõem o relatório.
2. Na faixa Início, selecione Editar, Localizar & Selecionar, Ir para Especial para abrir a caixa de diálogo Ir para Especial. Selecione Em Branco para selecionar apenas as células em branco.
3. Insira uma fórmula no estilo L1C1 para preencher a lacuna com a célula acima dela. Essa fórmula é `=L[-1]C`. Na interface com o usuário, digite um sinal de igual, pressione a tecla de seta para cima e, depois, a tecla Ctrl+Enter.
4. Selecione novamente todas as células na Coluna A que compõem o relatório. Esse passo é necessário porque o passo Colar Especial não funciona com seleções não-contíguas.
5. Copie as fórmulas na Coluna A e converta-as em valores escolhendo Área de Transferência, Colar, Colar Valores na guia Início.

Corrigir a visualização Estrutura de Tópicos no VBA requer menos passos. A lógica VBA equivalente é mostrada aqui:

1. Localize a última linha do relatório.
2. Insira a fórmula `=L[-1]C` nas células em branco em A.
3. Transforme essas fórmulas em valores. O código para fazer isso é mostrado a seguir:

```
Dim FinalReportRow as Long
    ' Preenche a visualização Estrutura de Tópicos na coluna A
    ' Procura a última linha na coluna B visto que muitas linhas na coluna A estão em branco
FinalReportRow = WSR.Cells(Rows.Count, 2).End(xlUp).Row
With Range("A3").Resize(FinalReportRow - 2, 1)
    With .SpecialCells(xlCellTypeBlanks)
        .FormulaR1C1 = "=R[-1]C"
    End With
    .Value = .Value
End With
```

Tratando a formatação final

O últimos passos para finalizar o relatório envolvem algumas tarefas básicas de formatação e, então, de adicionar os subtotais. Você pode aplicar negrito aos títulos e alinhá-los à direita na Linha 3. Configure as linhas 1 a 3 para que as três primeiras linhas sejam impressas em *todas* as páginas:

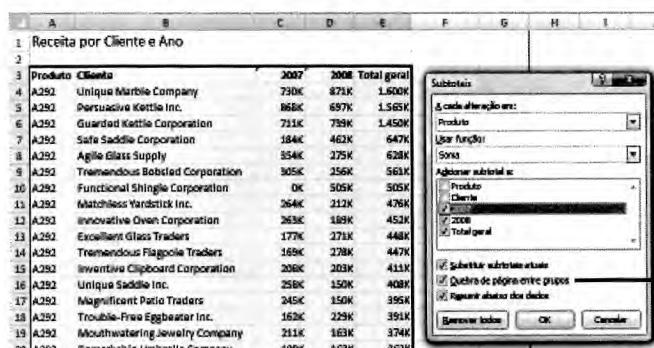
```
'Aplica uma formatação básica
'Autoajusta colunas, aplica negrito aos títulos, e os alinha à direita
Selection.Columns.AutoFit
Range("A3").EntireRow.Font.Bold = True
Range("A3").EntireRow.HorizontalAlignment = xlRight
Range("A3:B3").HorizontalAlignment = xlLeft

'Repete as linhas 1-3 na parte superior de cada página
WSR.PageSetup.PrintTitleRows = "$1:$3"
```

Adicionando subtotais

Os subtotais automáticos são um recurso poderoso localizado na faixa Dados. A Figura 13.14 mostra a caixa de diálogo Subtotais. Note a opção Quebra de Página Entre Grupos.

Se estivesse certo de que sempre teria dois anos e um total, o código para adicionar subtotais a cada grupo de linha de negócios seria o seguinte:

```
'Adiciona subtotais por produtos.
'Certifica-se de inserir uma quebra de página em todas as alterações de produtos
Selection.Subtotal GroupBy:=1, Function:=xlSum, TotalList:=Array(3, 4, 5),PageBreaks:=True
```

Figura 13.14
Utilize subtotais automáticos porque isso permite inserir uma quebra de página depois de cada produto. Utilizar esse recurso assegura que todos os gerentes de produtos tenham um relatório claro apenas com seus produtos.

Mas esse código falha se houver mais ou menos de três anos. A solução é utilizar este código complicado para criar dinamicamente uma lista das colunas para soma, com base no número de colunas no relatório:

```
Dim TotColumns()
Dim I as Integer
FinalCol = Cells(3, Columns.Count).End(xlToLeft).Column
ReDim Preserve TotColumns(1 To FinalCol - 2)
For i = 3 To FinalCol
    TotColumns(i - 2) = i
Next i
Selection.Subtotal GroupBy:=1, Function:=xlSum, TotalList:=TotColumns,_
     Replace:=True, PageBreaks:=True, SummaryBelowData:=True
```

Por fim, com a adição dos novos totais ao relatório, você precisa autoajustar novamente a coluna numérica com este código:

```
Dim GrandRow as Long
'Certifica-se de que as colunas têm uma largura suficiente para totais
GrandRow = Cells(Rows.Count, 1).End(xlUp).Row
Cells(3, 3).Resize(GrandRow - 2, FinalCol - 2).Columns.AutoFit
Cells(GrandRow, 3).Resize(1, FinalCol - 2).NumberFormat = "#,##0,K"
'Adiciona uma quebra de página antes da linha Total Geral, caso contrário
'o gerente de produto para a Linha final terá dois totais
WSR.HPageBreaks.Add Before:=Cells(GrandRow, 1)
```

Agrupando tudo

A Listagem 13.3 cria os relatórios do gerente de linha de produtos em alguns segundos.

Listagem 13.3 Código que produz o relatório de linha de produtos da Figura 13

```
Sub ProductLineReport()
    ' Produto e Cliente como Linha
    ' Anos definidos em Coluna
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long
    Dim TotColumns()

    Set WSD = Worksheets("TabelaDinâmica")

    Dim WSR As Worksheet
    Dim WBO As Workbook
    Dim WBN As Workbook
    Set WBO = ActiveWorkbook

    ' Exclui todas as tabelas dinâmicas anteriores
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("J1:Z1").EntireColumn.Clear

    ' Define a área de entrada e configura um cache dinâmico
    FinalRow = WSD.Cells(Application.Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Application.Columns.Count).End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, SourceData:=PRange.Address)

    ' Cria a tabela dinâmica a partir do cache dinâmico
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD.Cells(2, FinalCol + 2), TableName:="Tabela dinâmica1

    ' Desativa a atualização ao criar a tabela
    PT.ManualUpdate = True

    ' Configura os campos de linha
    PT.AddFields RowFields:=Array("Produto","Cliente"), ColumnFields:="Data"

    ' Configura os campos de dados
    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
    End With

    ' Certifica-se de fazer um layout de tabela em vez do novo layout compacto
    PT.RowAxisLayout xlTabularRow

    ' Calcula a tabela dinâmica
    PT.ManualUpdate = False
    PT.ManualUpdate = True

    ' Agrupa por ano
    WSD.Activate
    Cells(3, FinalCol + 4).Group Start:=True, End:=True, _
        Periods:=Array(False, False, False, False, False, False, True)

    ' Move Data para colunas
    PT.PivotFields("Data").Orientation = xlColumnField
    PT.PivotFields("Cliente").Orientation = xlRowField

    ' Formata os campos Receita
    PT.PivotFields("Soma de Receita").NumberFormat = "#,##0,K"

    ' Desativa os subtotais por produto
    PT.PivotFields("Produto").Subtotals(1) = True
```

```
    PT.PivotFields("Produto").Subtotals(1) = False
    PT.ColumnGrand = False

    ' Assegura que obtemos zeros em vez de lacunas na área de dados
    PT.NullString = "0"

    ' Classifica clientes em ordem decrescente por soma de receita
    PT.PivotFields("Cliente").AutoSort Order:=xlDescending,Field:="Soma de Receita"

    ' Calcula a tabela dinâmica
    PT.ManualUpdate = False
    PT.ManualUpdate = True

    ' Nesse ponto, os dados são semelhantes aos mostrados na Figura 13.13
    PT.TableRange2.Select
    ' Parada

    ' Cria uma nova pasta de trabalho em branco com uma planilha
    Set WBN = Workbooks.Add(xlWBATWorksheet)
    Set WSR = WBN.Worksheets(1)
    WSR.Name = "Report"
    ' Configura o título do relatório
    With WSR.[A1]
        .Value = "Receita por cliente e ano"
        .Font.Size = 14
    End With

    ' Copia os dados de tabela dinâmica para a linha 3 da planilha do relatório
    ' Usa Offset para eliminar a linha de título da tabela dinâmica
    PT.TableRange2.Offset(1, 0).Copy
    WSR.[A3].PasteSpecial Paste:=xlPasteValuesAndNumberFormats
    PT.TableRange2.Clear
    Set PTCache = Nothing

    ' Preenche a visualização Estrutura de Tópicos na coluna A
    ' Procura a última linha na coluna B visto que muitas linhas na coluna A estão em branco
    FinalReportRow = WSR.Range("B65536").End(xlUp).Row
    With Range("A3").Resize(FinalReportRow - 2, 1)
        With .SpecialCells(xlCellTypeBlanks)
            .FormulaR1C1 = "=R[-1]C"
        End With
        .Value = .Value
    End With

    ' Aplica uma formatação básica
    ' Autoajusta colunas, aplica negrito aos títulos e os alinha à direita
    Selection.Columns.AutoFit
    Range("A3").EntireRow.Font.Bold = True
    Range("A3").EntireRow.HorizontalAlignment = xlRight
    Range("A3:B3").HorizontalAlignment = xlLeft

    ' Repete as linhas 1-3 na parte superior de cada página
    WSR.PageSetup.PrintTitleRows = "$1:$3"

    ' Soma subtotais
    FinalCol = Cells(3, 255).End(xlToLeft).Column
    ReDim Preserve TotColumns(1 To FinalCol - 2)
    For i = 3 To FinalCol
        TotColumns(i - 2) = i
    Next i
    Selection.Subtotal GroupBy:=1, Function:=xlSum,TotalList:=TotColumns, Replace:=True,PageBreaks:=True, _
        SummaryBelowData:=True

    ' Certifica-se de que as colunas têm uma largura suficiente para totais
    GrandRow = Cells(Rows.Count, 1).End(xlUp).Row
    Cells(3, 3).Resize(GrandRow - 2, FinalCol - 2).Columns.AutoFit
    Cells(GrandRow, 3).Resize(1, FinalCol - 2).NumberFormat = "#,##0,K"
    ' Adiciona uma quebra de página antes da linha Total Geral, caso contrário
    ' o gerente de produto para a Linha final obterá dois totais
    WSR.HPageBreaks.Add Before:=Cells(GrandRow, 1)

End Sub
```

A Figura 13.15 mostra o relatório produzido por esse código.

A	B	C	D	E
Receita por Cliente e Ano				
Produto	Cliente	2007	2008	Total geral
A292	Unique Marble Company	730K	871K	1.600K
A292	Persuasive Kettle Inc.	868K	697K	1.565K
A292	Guarded Kettle Corporation	711K	739K	1.450K
A292	Safe Saddle Corporation	184K	462K	647K
A292	Agile Glass Supply	354K	275K	628K
A292	Tremendous Bobsled Corporation	305K	256K	561K
A292	Functional Shingle Corporation	0K	505K	505K
A292	Matchless Yardstick Inc.	264K	212K	476K
A292	Innovative Oven Corporation	263K	189K	452K
A292	Excellent Glass Traders	177K	271K	448K
A292	Tremendous Flagpole Traders	169K	278K	447K
A292	Inventive Clipboard Corporation	208K	203K	411K
A292	Unique Saddle Inc.	258K	150K	408K
A292	Magnificent Patio Traders	245K	150K	395K
A292	Trouble-Free Eggbeater Inc.	162K	229K	391K
A292	Mouthwatering Jewelry Company	211K	163K	374K
A292	Remarkable Umbrella Company	199K	163K	363K
A292	Mouthwatering Tripod Corporation	138K	199K	337K
A292	Vibrant Tripod Corporation	124K	194K	318K
A292	Enhanced Toothpick Corporation	148K	145K	293K
A292	Persuasive Yardstick Corporation	92K	176K	268K
A292 Total		5.811K	6.526K	12.338K
B722	Unique Marble Company	755K	827K	1.582K
B722	Guarded Kettle Corporation	843K	562K	1.405K

Figura 13.15
Converter 50 mil linhas de dados transacionais para esse relatório útil leva menos de alguns segundos se você utilizar o código que produziu esse exemplo. Sem as tabelas dinâmicas, o código seria muito mais complexo.

Abordando questões com dois ou mais campos de dados

Até agora, você construiu alguns relatórios de resumo poderosos, mas tocou apenas em uma parte dos poderosos recursos disponíveis em tabelas dinâmicas. O exemplo anterior produziu um relatório que tinha apenas um campo de dados. É possível ter múltiplos campos na seção □ Valores de um relatório dinâmico. Os dados nesse exemplo incluem não só receita, mas também uma contagem de clientes.

Quando tiver dois ou mais campos Dados, você pode escolher colocar os campos Dados em um de quatro locais. Por padrão, o Excel cria o relatório dinâmico com o campo Dados como o campo Coluna mais interno. Em geral, é preferível ter o campo Dados como o campo Linha mais externo.

Quando uma tabela dinâmica tiver mais de um campo de dados, você têm um nome de campo virtual □ Valores nas áreas da Lista de Campos da Tabela Dinâmica. No VBA, esse campo virtual equivalente é chamado Dados.

Para organizar os campos de modo que o campo de dados seja a linha mais interna, como mostrado na Figura 13.16, utilize esta linha `AddFields`:

```
PT.AddFields RowFields:=Array("Produto", "Dados")
```

Produto	Dados	Total
A292	Soma de Receita	12.337.778
	Soma de Lucro	5936710
	Contagem de Clientes	998
B722	Soma de Receita	12.683.061
	Soma de Lucro	6067502
	Contagem de Clientes	978
C409	Soma de Receita	14.101.763
	Soma de Lucro	6765174
	Contagem de Clientes	995
D625	Soma de Receita	14.569.960
	Soma de Lucro	6972477
	Contagem de Clientes	982
E438	Soma de Receita	17.411.966
	Soma de Lucro	8354215
	Contagem de Clientes	1035
Total Soma de Receita		71.104.528
Total Soma de Lucro		34096078
Total Contagem de Clientes		4988

Figura 13.16
Adicionar dados, como o campo Linha interno, apresenta essa visualização.

Se mover o campo Dados como o primeiro campo Linha, você terá a visualização mostrada na Figura 13.17. Nessa versão, Soma Total da Receita aparece a 10 linhas de todos os outros campos de receita. A visualização mostrada na Figura 13.17 usaria este código:

```
PT.AddFields RowFields:=Array("Dados", "Produto")
```

Uma visualização que faria sentido teria Dados como o único campo Coluna:

```
PT.AddFields RowFields:="Produto", ColumnFields:="Dados"
```

Dados	Produto	Total
Soma de Receita	A292	12.337.778
	B722	12.683.061
	C409	14.101.763
	D625	14.569.960
	E438	17.411.966
Soma de Lucro	A292	5936710
	B722	6067502
	C409	6765174
	D625	6972477
	E438	8354215
Contagem de Clientes	A292	998
	B722	978
	C409	995
	D625	982
	E438	1035
Total Soma de Receita		71.104.528
Total Soma de Lucro		34096078
Total Contagem de Clientes		4988

Figura 13.17
Mova o campo de dados para um local antes do campo Produto e verá também essa visualização esquisita dos dados. Realmente odeio quando a linha *Soma Total* da Receita fica muito longe dos subtotais de Receita individuais.

Depois de adicionar um campo Coluna chamado Dados, avance para definir três campos de dados:

```
'Configura os campos de dados
With PT.PivotFields("Receita")
    .Orientation = xlDataField
    .Function = xlSum
    .Position = 1
    .NumberFormat = "#,##0,K"
End With

With PT.PivotFields("Lucro")
    .Orientation = xlDataField
    .Function = xlSum
    .Position = 2
    .NumberFormat = "#,##0"
End With

With PT.PivotFields("Cliente")
    .Orientation = xlDataField
    .Function = xlCount
    .Position = 3
    .NumberFormat = "#,##0"
End With
```

A Figura 3.18 mostra o relatório produzido pelo código anterior.

	Dados		
Produto	Soma de Receita	Soma de Lucro	Contagem de Clientes
A292	12.337.778	5936710	998
B722	12.683.061	6067502	978
C409	14.101.763	6765174	995
D625	14.569.960	6972477	982
E438	17.411.966	8354215	1035
Total geral	71.104.528	34096078	4988

Figura 13.18
Ao mover o campo de dados para o campo Coluna, obtemos um relatório com uma aparência relativamente normal.

Campo de dados calculados

As tabelas dinâmicas oferecem dois tipos de fórmulas. O tipo mais útil define uma fórmula para um campo calculado. Isso insere um novo campo na tabela dinâmica. Os cálculos para os campos calculados são sempre feitos no nível de resumo. Se você definir um campo calculado para o preço médio como Receita dividida por Unidades Vendidas, o Excel primeiro adicionará a receita total e a quantidade total e, depois, fará a divisão desses totais para obter o resultado. Em muitos casos, isso é exatamente o que você precisa. Se seu cálculo não seguir a lei associativa de matemática, ele pode não funcionar da maneira esperada.

Para configurar um campo calculado, utilize o método `Add` com o objeto `CalculatedFields`. É preciso especificar um nome de campo e uma fórmula. Note que, se você criar um campo chamado Percentagem de Lucro, a tabela dinâmica padrão produzirá um campo chamado Soma da Percentagem de Lucro. Esse título é confuso e claramente ridículo. A solução é utilizar a propriedade `Name` ao definir o campo de dados para substituir Soma da Percentagem de Lucro por algo como Lucro Pct. Observe que esse nome deve diferir do nome do campo calculado.

A Listagem 13.4 produz o relatório mostrado na Figura 13.19.

Figura 13.19
A dimensão de dados virtuais contém dois campos de seu conjunto de dados, mais um cálculo. Ele é mostrado ao longo da área de coluna do relatório.

	Dados		
Produto	Soma de Receita	Soma de Lucro	Lucro Pct
A292	12.337.778	5.936.710	48,1%
B722	12.683.061	6.067.502	47,8%
C409	14.101.763	6.765.174	48,0%
D625	14.559.023	6.967.788	47,9%
E438	17.411.966	8.354.215	48,0%
Total geral	71.093.591	34.091.389	48,0%

Listagem 13.4 Código que calcula uma Percentagem de Lucro como um segundo campo de dados

```
Sub CalculatedField()
    'Listagem 13.4
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long

    Set WSD = Worksheets("TabelaDinâmica")
    Dim WSR As Worksheet
    Dim WBO As Workbook
    Dim WBN As Workbook
    Set WBO = ActiveWorkbook

    ' Exclui todas as tabelas dinâmicas anteriores
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("J1:Z1").EntireColumn.Clear

    ' Define a área de entrada e configura um cache dinâmico
    FinalRow = WSD.Cells(Application.Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Application.Columns.Count).End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, SourceData:=PRange.Address)

    ' Cria a tabela dinâmica a partir do cache dinâmico
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD.Cells(2, FinalCol + 2), TableName:="PivotTable1")

    ' Desativa a atualização ao criar a tabela
    PT.ManualUpdate = True

    ' Configura os campos de linha
    PT.AddFields RowFields:="Produto", ColumnFields:="Dados

    ' Define campos calculados
    PT.CalculatedFields.Add Name:="PorcentagemLucro", Formula:="=Lucro/Receita"

    ' Configura os campos de dados
    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
        .NumberFormat = "#,##0"
    End With
```

```
    With PT.PivotFields("Lucro")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 2
        .NumberFormat = "#,##0"
    End With

    With PT.PivotFields("PorcentagemLucro")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 3
        .NumberFormat = "#0.0%"
        .Name = "Lucro Pct"
    End With

    ' Assegura que obtemos zeros em vez de lacunas na área de dados
    PT.NullString = "0"

    ' Calcula a tabela dinâmica
    PT.ManualUpdate = False
    PT.ManualUpdate = True

    WSD.Activate
    Range("J2").Select

End Sub
```

Estudo de caso

Itens calculados

Suponha que na sua empresa um gerente seja responsável pelos produtos A292 e C409. A idéia por trás de um item calculado é que você pode definir um novo item ao longo do campo Produto para calcular o total desses dois itens. A Listagem 13.5 produz o relatório mostrado na Figura 13.20.

Figura 13.20
A menos que você adore declarar repetidamente números para Seguradoras e Bolsas de Valores, evite utilizar itens calculados.

Soma de Receita	
Produto	Total
A292	12.337.778
C409	14.101.763
MinhaDivisão	26.439.541
B722	12.683.061
D625	14.559.023
E438	17.411.966
Total geral	97.533.132

Listagem 13.5 Código que insere um novo item na dimensão de produto

```
Sub CalcItemsProblem()
    ' Listagem 13.5
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long

    Set WSD = Worksheets("TabelaDinâmica")
    Dim WSR As Worksheet

    ' Exclui todas as tabelas dinâmicas anteriores
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("J1:Z1").EntireColumn.Clear

    ' Define a área de entrada e configura um cache dinâmico
    FinalRow = WSD.Cells(Application.Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Application.Columns.Count).End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, SourceData:=PRange.Address)
```

```
    ' Cria a tabela dinâmica do cache dinâmico
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD.Cells(2, FinalCol + 2), TableName:="Tabela _
        dinâmica1")

    ' Desativa a atualização ao criar a tabela
    PT.ManualUpdate = True

    ' Configura os campos de linha
    PT.AddFields RowFields:="Produto"

    ' Define o item calculado ao longo da dimensão de produto
PT.PivotFields("Produto").CalculatedItems.Add "MinhaDivisão", "='A292'+'C409'"
    ' Seqüencia novamente para que o relatório tenha A292 e C409 primeiro
    PT.PivotFields("Produto").PivotItems("A292").Position = 1
    PT.PivotFields("Produto").PivotItems("C409").Position = 2
    PT.PivotFields("Produto").PivotItems("MinhaDivisão").Position = 3

    ' Configura os campos de dados
    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
        .NumberFormat = "#,##0"
    End With

    ' Assegura que obtemos zeros em vez de lacunas na área de dados
    PT.NullString = "0"

    ' Calcula a tabela dinâmica
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    WSD.Activate
    Range("J2").Select

End Sub
```

Examine atentamente os resultados mostrados na Figura 13.20. O cálculo de MinhaDivisão está correto. Os US$ 26 milhões aproximados para MinhaDivisão é a soma de US$ 12 milhões e US$ 14 milhões de C409. Mas o total geral deve ser de aproximadamente US$ 71 milhões. Em vez disso, o Excel oferece uma soma total de US$ 97 milhões. A receita total da empresa aumentou em US$ 26 milhões. O Excel dará o total geral errado quando um campo contiver tanto itens regulares como calculados. O único método plausível para lidar com essa situação é tentar ocultar os produtos que compõem MinhaDivisão:

```
With PT.PivotFields("Produto")
    .PivotItems("A292").Visible = False
    .PivotItems("C409").Visible = False
End With
```

A Figura 13.21 mostra os resultados.

Soma de Receita	
Produto	Total
MinhaDivisão	26.439.541
B722	12.683.061
D625	14.559.023
E438	17.411.966
Total geral	71.093.591

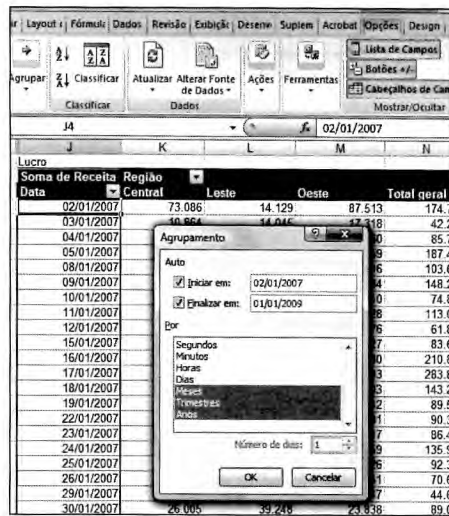
Figura 13.21
Depois que os componentes do item MinhaDivisão são ocultados, a receita total da empresa estará novamente correta. Mas seria mais fácil adicionar um novo campo aos dados originais com um campo Divisão.

Resumindo campos Data com agrupamento

Com os dados transacionais, muitas vezes você encontra os resumos baseados em data com uma linha por dia. Embora os dados diários possam ser úteis para gerentes de fábrica, muitas pessoas na empresa querem ver os totais por mês ou por trimestre e ano.

A ótima notícia é que o Excel trata a sumarização de datas em uma tabela dinâmica com facilidade. Para alguém que nunca teve de utilizar a misteriosa fórmula `=A2-DAY(A2)+1` para mudar datas diárias para datas mensais, você apreciará a simplicidade com que pode agrupar os dados transacionais em meses ou trimestres.

Na Figura 13.22, selecione uma célula que contenha uma data. Na faixa Opções, escolha Agrupar Campo. Na caixa de diálogo Agrupamento, escolha agrupar por Meses, Trimestres e Anos.

Figura 13.22
Utilize a caixa de diálogo Agrupamento para mudar um relatório menos significativo de datas diárias para um resumo mensal, trimestral e anual.

> **ATENÇÃO**
>
> Nunca escolha agrupar somente por meses, sem incluir os anos. Se fizer isso, o Excel combinará janeiro deste ano e janeiro do ano passado em um único item chamado Janeiro. Embora isso seja ótimo para análise de sazonalidade, raramente é o que você quer em um resumo. Sempre escolha Anos e Meses na caixa de diálogo Agrupamento.

Entendendo o método Group no VBA

Criar um grupo com o VBA é um pouco estranho. O método `.Group` pode ser aplicado a apenas uma única célula na tabela dinâmica e essa célula deve conter uma data ou rótulo de campo Data. Esse é o primeiro exemplo neste capítulo onde você deve permitir ao VBA calcular um resultado intermediário de tabela dinâmica.

Você deve definir uma tabela dinâmica com Data de Fatura no campo Linha. Desative `ManualCalculation` para permitir que o campo Data seja feito. Você pode então utilizar a propriedade `LabelRange` para localizar o rótulo de data e aplicar o método `.Group`.

```
PT.PivotFields("Data").LabelRange.Group
```

Para especificar como agrupar o campo Data, você tem de passar um array de sete valores `True/False`. O primeiro valor corresponde à seleção de Segundos na caixa de diálogo Agrupamento. O próximo valor corresponde a Minutos, e então Horas, Dias, Meses, Trimestres e Anos. Nesse exemplo, você quer agrupar por meses, trimestres e anos; portanto, o argumento Períodos é conforme o mostrado a seguir:

```
Periods:=Array(False, False, False, False, True, True, True)
```

A Figura 13.23 mostra o resultado da Listagem 13.6.

Figura 13.23
O campo Data agora é composto de três campos na tabela dinâmica, representando ano, trimestre e mês.

Soma de Receita			Região			
Anos	Trimestres	Data	Central	Leste	Oeste	Total geral
2007	Trim1	jan	934.996	759.812	925.754	2.620.562
		fev	940.552	734.213	743.100	2.417.865
		mar	847.410	945.557	849.296	2.642.263
	Trim2	abr	1.041.557	1.157.651	904.315	3.103.523
		mai	1.126.004	1.070.956	1.036.049	3.233.009
		jun	985.251	926.346	730.321	2.641.918
	Trim3	jul	789.391	1.028.495	844.630	2.662.516
		ago	985.769	1.297.610	1.008.813	3.292.192
		set	900.029	1.022.965	1.086.848	3.009.842
	Trim4	out	931.141	930.508	825.500	2.687.149
		nov	903.724	1.044.264	793.992	2.741.980
		dez	1.226.581	912.991	1.016.044	3.155.616
2008	Trim1	jan	813.970	1.091.723	1.191.212	3.096.905
		fev	771.282	966.591	911.543	2.649.416
		mar	908.217	1.016.731	940.637	2.865.585
	Trim2	abr	778.236	1.139.337	977.767	2.895.340
		mai	929.291	1.131.145	1.170.395	3.230.831
		jun	858.502	875.094	1.034.287	2.767.883
	Trim3	jul	1.250.464	1.155.336	1.231.973	3.637.773
		ago	810.882	1.108.519	1.054.510	2.973.911
		set	1.033.327	1.138.887	1.036.654	3.208.868
	Trim4	out	1.239.788	1.110.850	1.133.920	3.484.558
		nov	1.125.964	695.423	918.852	2.740.239
		dez	860.453	1.070.421	1.413.910	3.344.784
Total geral			22.992.781	24.331.425	23.780.322	71.104.528

Lista de campos da tabela dinâmica
Escolha os campos para adicionar ao relatório:
- [x] Região
- [ ] Produto
- [x] Data
- [ ] Cliente
- [ ] Quantidade
- [x] Receita
- [ ] Custos de Vendas
- [ ] Lucro
- [x] Trimestres
- [x] Anos

Arraste os campos entre as áreas abaixo:
Filtro de Relatório | Rótulos de Coluna: Região
Rótulos de Linha: Anos, Trimestres, Data | Valores: Soma de Receita
Adiar Atualização do Layout | Atualizar

Listagem 13.6 Código que utiliza o recurso Agrupar para rolar de datas diárias até datas mensais

```
Sub ReportByMonth()
    ' Listagem 13.6
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long

    Set WSD = Worksheets("TabelaDinâmica")
    Dim WSR As Worksheet

    ' Exclui todas as tabelas dinâmicas anteriores
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("R1:AZ1").EntireColumn.Clear

    ' Define a área de entrada e configura um cache dinâmico
    FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Columns.Count).End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, SourceData:=PRange.Address)

    ' Cria a tabela dinâmica a partir do cache dinâmico
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD.Cells(2, FinalCol + 2), _
        TableName:="Tabela dinâmica1")

    ' Desativa a atualização ao criar a tabela
    PT.ManualUpdate = True

    ' Configura os campos de linha
    PT.AddFields RowFields:="Data", ColumnFields:="Região"

    ' Configura os campos de dados
    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
        .NumberFormat = "#,##0"
    End With

    ' Assegura que obtemos zeros em vez de lacunas na área de dados
    PT.NullString = "0"

    ' Calcula a tabela dinâmica para permitir que o rótulo de data seja elaborado
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    WSD.Activate

    ' Agrupa DataEntrega por mês, trimestre, ano
    PT.PivotFields("Data").LabelRange.GroupStart:=True, End:=True,_
        Periods:=Array(False, False, False, False, True, True, True)

    ' Calcula a tabela dinâmica
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    WSD.Activate
    Range("R1").Select

End Sub
```

Agrupamento por semana

Você provavelmente notou que o Excel permite agrupar por dia, mês, trimestre e ano. Não há agrupamento-padrão por semana, mas é possível definir um grupo que reúne grupos de sete dias.

Por padrão, o Excel inicia a semana com base na primeira data localizada nos dados. Isso significa que a semana-padrão teria de executar a partir de terça-feira, 2 de janeiro de 2007, a segunda-feira, 8 de janeiro de 2007. Você pode sobrescrever isso alterando o parâmetro `Start` de `True` para uma data real.

Imaginar a primeira data correta exige algumas funções básicas do Excel. Você pode utilizar a função `MIN` para localizar as primeiras datas da tabela dinâmica com este código:

```
FirstDate = Application.WorksheetFunction.Min(PT.PivotFields("Data").DataRange)
```

Depois, é possível então utilizar a função `WeekDay` para determinar em quantos dias ajustar a data inicial:

```
WhichDay = Application.WorksheetFunction.Weekday(FirstDate, 3)
StartDate = FirstDate - WhichDay
```

Depois de determinar a data em que a primeira semana deve iniciar, utilize o código seguinte para agrupar o campo por grupos de sete dias:

```
PT.PivotFields("Data").LabelRange.GroupStart:=StartDate, _
    End:=True, By:=7,Periods:=Array(False, False, False, True, False, False, False)
```

ATENÇÃO

Há uma limitação para agrupar por semana. Ao fazer isso, você não pode, ao mesmo tempo, agrupar por qualquer outra medida. Por exemplo, não é válido agrupar por semana e trimestre.

A Listagem 13.7 cria o relatório mostrado na Figura 13.24.

Figura 13.24
Utilize a configuração Número de Dias para agrupar por semana.

Soma de Receita	Região			
Data	Central	Leste	Oeste	Total geral
01/01/2007 - 07/01/2007	183.322	157.365	149.450	490.137
08/01/2007 - 14/01/2007	167.188	141.518	193.064	501.770
15/01/2007 - 21/01/2007	322.696	257.753	230.795	811.244
22/01/2007 - 28/01/2007	177.386	107.575	190.874	475.835
29/01/2007 - 04/02/2007	187.357	133.011	197.142	517.510
05/02/2007 - 11/02/2007	147.318	242.677	212.607	602.602
12/02/2007 - 18/02/2007	258.601	143.336	192.265	594.202
19/02/2007 - 25/02/2007	234.428	216.122	135.000	585.550
26/02/2007 - 04/03/2007	329.825	102.274	200.663	632.762
05/03/2007 - 11/03/2007	255.519	157.667	247.795	660.981
12/03/2007 - 18/03/2007	209.045	149.134	280.825	639.004
19/03/2007 - 25/03/2007	81.754	288.900	183.425	554.079
26/03/2007 - 01/04/2007	168.519	342.250	93.308	604.077
02/04/2007 - 08/04/2007	270.718	363.424	217.197	851.339

Listagem 13.7 Código que utiliza o recurso Agrupamento para rolar por datas diárias até datas semanais

```
Sub ReportByWeek()
    ' Listagem 13.7
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long

    Set WSD = Worksheets("TabelaDinâmica")
    Dim WSR As Worksheet

    ' Exclui todas as tabelas dinâmicas anteriores
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("J1:Z1").EntireColumn.Clear

    ' Define a área de entrada e configura um cache dinâmico
    FinalRow = WSD.Cells(Application.Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Application.Columns.Count).End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, SourceData:=PRange.Address)

    ' Cria a tabela dinâmica a partir do cache dinâmico
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD.Cells(2, FinalCol + 2), TableName:="Tabela _
        dinâmica1")

    ' Desativa a atualização ao criar a tabela
    PT.ManualUpdate = True
```

```
    ' Configura os campos de linha
    PT.AddFields RowFields:="Data", ColumnFields:="Região"

    ' Configura os campos de dados
    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
        .NumberFormat = "#,##0"
    End With

    ' Assegura que obtemos zeros em vez de lacunas na área de dados
    PT.NullString = "0"

    ' Calcula a tabela dinâmica para permitir que o rótulo de data seja elaborado
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    WSD.Activate

    ' Agrupa data por semana.
    'Descobre a primeira segunda-feira antes da data mínima
    FirstDate = Application.WorksheetFunction.Min(PT.PivotFields("Data").DataRange)
    WhichDay = Application.WorksheetFunction.Weekday(FirstDate, 3)
    StartDate = FirstDate - WhichDay
    PT.PivotFields("Data").LabelRange.GroupStart:=StartDate, End:=True, By:=7,_
        Periods:=Array(False, False, False, True, False, False, False)

    ' Calcula a tabela dinâmica
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    WSD.Activate
    Range("J2").Select

End Sub
```

Medindo o tempo de espera do pedido pelo agrupamento de dois campos de data

A partir da última seção, lembre-se de como o Excel adiciona nomes de campo quando se agrupa por mês e ano. Medidas menos agregadas, como mês, herdam o nome do campo original. Medidas mais agregadas, como anos, recebem o nome do agrupamento.

Sua fábrica talvez esteja interessada em saber com quanto tempo de antecedência os pedidos são recebidos. Se pudesse dispor de 12 semanas para obter os componentes, gostaria que todos os pedidos fossem feitos com 13 semanas de antecedência. Quando isso não acontece, é crucial que você tenha um excelente sistema instalado para prever exatamente os pedidos.

Se puder inserir um campo `DataPedido` aos dados transacionais, você poderá criar uma tabela para mostrar quanta receita é recebida *x* meses de antecedência da data de entrega.

Siga estes passos para configurar uma potencial anomalia interessante:

1. Crie uma tabela dinâmica com `DataEntrega` na área Coluna, `DataPedido` na área Linha e `Receita` na área Dados.
2. Permita que a tabela dinâmica calcule.
3. Agrupe DataEntrega por mês e ano. Isso cria os campos chamados DataEntrega e Ano.
4. Permita que a tabela dinâmica calcule. Se tentar agrupar o campo DataPedido antes de calcular os resultados do passo 3, você obterá um erro.
5. Agrupe DataPedido por mês e ano. Isso cria um campo chamado DataPedido com dados por mês. O agrupamento de DataPedido por ano também tenderia a ser chamado Ano, mas em vez disso o Excel o chama de Ano2. A menos que você saiba que o código agrupou DataEntrega antes de DataPedido, nunca saberia que Ano2 se referia à data do pedido e não à data de entrega.

Versões do Excel desde 2000 lidam corretamente com o segundo conjunto de campos de Data agrupados, mudando o nome de campo para Ano2 em vez de ter um segundo Ano.

A Listagem 13.8 cria o relatório mostrado na Figura 13.25.

Listagem 13.8 Código utilizado para criar o relatório de tempo de espera do pedido

```
Sub MeasureLeadtime2007()
    ' Listagem 13.8
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long

    Set WSD = Worksheets("TempoEspera")
    Dim WSR As Worksheet

    ' Exclui todas as tabelas dinâmicas anteriores
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT

    ' Define a área de entrada e configura um cache dinâmico
    FinalRow = WSD.Cells(Application.Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Application.Columns.Count).End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, SourceData:=PRange.Address)

    ' Cria a tabela dinâmica a partir do cache dinâmico
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD.Cells(2, FinalCol + 2), TableName:=" Pivot _
        Table1")

    ' Desativa a atualização ao criar a tabela
    PT.ManualUpdate = True

    ' Configura os campos de linha
    PT.AddFields RowFields:="DataPedido", ColumnFields:="DataEntrega"

    ' Configura os campos de dados
    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
        .NumberFormat = "#,##0"
    End With

    ' Assegura que obtemos zeros em vez de lacunas na área de dados
    PT.NullString = "0"

    ' Calcula a tabela dinâmica para permitir que o rótulo de data seja elaborado
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    WSD.Activate

    ' Agrupa ShipDate por mês e ano
    PT.PivotFields("DataEntrega").LabelRange.Group Start:=True, End:=True, _
         Periods:=Array(False, False, False, False, True, False, True)

    ' Calcula a tabela dinâmica para permitir que o rótulo de data seja elaborado
    PT.ManualUpdate = False
    PT.ManualUpdate = True

    ' Agrupa OrderDate por mês e ano
    PT.PivotFields("DataPedido").LabelRange.Group Start:=True, _
         End:=True,Periods:=Array(False, False, False, False, True, False, True)

    ' Calcula a tabela dinâmica
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    WSD.Activate
    Range("K2").Select

End Sub
```

Soma de Receita		Anos / DataEntrega				
		2007				
Anos2	DataPedido	jan	fev	mar	abr	mai
2006	ago	316.572	0	0	0	
	set	646.490	182.174	0	0	
	out	499.989	597.329	214.511	0	
	nov	414.669	491.187	526.299	414.122	
	dez	645.649	489.733	577.506	759.588	
2007	jan	97.193	532.951	553.808	634.511	
	fev	0	124.491	648.716	541.929	
	mar	0	0	121.423	613.597	
	abr	0	0	0	139.776	
	mai	0	0	0	0	

Figura 13.25
Esse relatório de tempo de preparação de pedidos mostra que é melhor você ter um excelente sistema de planejamento de vendas e operações.

Utilizando técnicas avançadas de tabela dinâmica

Você pode ser um profissional de tabela dinâmica e nunca se deparar com algumas técnicas realmente avançadas disponíveis com as tabelas dinâmicas. As seções a seguir discutem essas técnicas.

Utilizando AutoShow para produzir resumos executivos

Se você estiver projetando um utilitário de dashboard (painel) executivo, talvez queira chamar a atenção para os cinco maiores clientes.

Como na opção AutoClassificar, você pode ser um profissional de tabela dinâmica e encontrar por acaso o recurso AutoShow do Excel. Essa configuração permite selecionar os *n* registros superiores ou inferiores com base em qualquer campo de dados no relatório.

O código para utilizar AutoShow no VBA utiliza o método `.AutoShow`:

```
'Mostra apenas os 5 maiores clientes
PT.PivotFields("Cliente").AutoShow Top:=xlAutomatic, Range:=xlTop, Count:=5, Field:= "Soma de Receita"
```

Quando você cria um relatório utilizando o método `.AutoShow`, é muito útil copiar os dados e voltar para o relatório dinâmico original para obter os totais de todos os mercados. No código mostrado na Listagem 13.9, isso é alcançado pela remoção do campo Cliente da tabela dinâmica e cópia do total para o relatório. A Listagem 13.9 produz o relatório mostrado na Figura 13.26.

	A	B	C	D	E	F	G
1	Os 5 maiores clientes						
2							
3	Cliente	A292	B722	C409	D625	E438	Total geral
4	Guarded Kettle Corporation	1.450.110	1.404.742	1.889.149	1.842.751	2.302.023	8.888.775
5	Unique Marble Company	1.600.347	1.581.665	1.765.305	1.707.140	2.179.242	8.833.699
6	Persuasive Kettle Inc.	1.565.368	1.385.296	1.443.434	1.584.759	2.030.578	8.009.435
7	Safe Saddle Corporation	646.559	857.573	730.463	1.038.371	1.053.369	4.326.335
8	Tremendous Bobsled Corporation	560.759	711.826	877.247	802.303	1.095.329	4.047.464
9	Total dos 5 Maiores	5.823.143	5.941.102	6.705.598	6.975.324	8.660.541	34.105.708
10							
11	Total da empresa	12.337.778	12.683.061	14.101.763	14.569.960	17.411.966	71.104.528

Figura 13.26
O relatório dos cinco maiores clientes contém duas tabelas dinâmicas.

Listagem 13.9 Código utilizado para criar o relatório dos cinco maiores clientes

```
Sub Top5Customers()
    ' Listagem 13.9
    ' Produz um relatório dos 5 maiores clientes
    Dim WSD As Worksheet
    Dim WSR As Worksheet
    Dim WBN As Workbook
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long
    Set WSD = Worksheets("TabelaDinamica")

    ' Exclui todas as tabelas dinâmicas anteriores
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("J1:Z1").EntireColumn.Clear

    ' Define a área de entrada e configura um cache dinâmico
    FinalRow = WSD.Cells(Application.Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Application.Columns.Count).End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, SourceData:=PRange.Address)
```

```
    ' Cria a tabela dinâmica a partir do cache dinâmico
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD.Cells(2, FinalCol + 2), TableName:="PivotTable1")

    ' Desativa a atualização ao criar a tabela
    PT.ManualUpdate = True

    ' Configura os campos de linha
    PT.AddFields RowFields:="Cliente", ColumnFields:="Produto"

    ' Configura os campos de dados
    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
        .NumberFormat = "#,##0"
        .Name = "Receita Total"
    End With

    ' Assegura que obtemos zeros em vez de lacunas na área de dados
    PT.NullString = "0"

    ' Classifica clientes em ordem decrescente por soma de receita
    PT.PivotFields("Cliente").AutoSort Order:=xlDescending,Field:="Receita Total"

    ' Mostra somente os 5 maiores clientes
    PT.PivotFields("Cliente").AutoShow Type:=xlAutomatic, Range:=xlTop, _

        Count:=5, Field:="Receita Total"

    ' Calcula a tabela dinâmica para permitir que o rótulo de data seja elaborado
    PT.ManualUpdate = False
    PT.ManualUpdate = True

    ' Cria uma nova pasta de trabalho em branco com uma planilha
    Set WBN = Workbooks.Add(xlWBATWorksheet)
    Set WSR = WBN.Worksheets(1)
    WSR.Name = "Relatório"
    ' Configura título para o relatório
    With WSR.[A1]
        .Value = "5 maiores clientes"
        .Font.Size = 14
    End With

    ' Copia os dados da tabela dinâmica para a linha 3 da planilha de relatório
    ' Utiliza deslocamento para eliminar a linha de título da tabela dinâmica
    PT.TableRange2.Offset(1, 0).Copy
    WSR.[A3].PasteSpecial Paste:=xlPasteValuesAndNumberFormats
    LastRow = WSR.Cells(Rows.Count, 1).End(xlUp).Row
    WSR.Cells(LastRow, 1).Value = "Total 5 Maiores"

    ' Volta à tabela dinâmica para obter totais sem AutoShow
    PT.PivotFields("Cliente").Orientation = xlHidden
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    PT.TableRange2.Offset(2, 0).Copy
    WSR.Cells(LastRow + 2, 1).PasteSpecial Paste:=xlPasteValuesAndNumberFormats
    WSR.Cells(LastRow + 2, 1).Value = "Total da Empresa"

    ' Limpa a tabela dinâmica
    PT.TableRange2.Clear
    Set PTCache = Nothing

    ' Aplica uma formatação básica.
    ' Autoajusta colunas, aplica negrito aos títulos e os alinha à direita
    WSR.Range(WSR.Range("A3"), WSR.Cells(LastRow + 2, 6)).Columns.AutoFit
    Range("A3").EntireRow.Font.Bold = True
    Range("A3").EntireRow.HorizontalAlignment = xlRight
    Range("A3").HorizontalAlignment = xlLeft

    Range("A2").Select
    MsgBox "Relatório CEO foi criado"
End Sub
```

O relatório dos cinco maiores clientes *realmente* contém dois instantâneos de uma tabela dinâmica. Depois de utilizar o recurso AutoShow para obter os cinco maiores mercados com seus respectivos totais, a macro voltou à tabela dinâmica, removeu a opção AutoShow e obteve o total de todos clientes para produzir a linha Total da Empresa.

Utilizando `ShowDetail` para filtrar um recordset

Considere qualquer tabela dinâmica na interface com o usuário do Excel. Dê um clique duplo em qualquer número da tabela. O Excel irá inserir uma nova planilha na pasta de trabalho e copiar todos os registros de fonte que representam esse número. Na interface com o usuário do Excel, essa é uma excelente maneira de realizar uma consulta drill-down em um conjunto de dados.

A propriedade VBA equivalente é `ShowDetail`. Configurando essa propriedade como `True` para qualquer célula na tabela dinâmica, você gera uma nova planilha com todos os registros que compõem essa célula:

```
PT.TableRange2.Offset(2, 1).Resize(1, 1).ShowDetail = True
```

A Listagem 13.10 produz uma tabela dinâmica com a receita total para os três maiores clientes e `ShowDetail` para cada uma daquelas lojas. Trata-se de um método alternativo para utilizar o relatório Filtro Avançado. Os resultados dessa macro são três novas planilhas. A Figura 13.27 mostra a primeira planilha criada.

Figura 13.27
Os aplicativos de tabela dinâmica são incrivelmente diversos. Essa macro criou uma tabela dinâmica das três maiores lojas e, então, utilizou a propriedade `ShowDetail` para recuperar os registros para cada uma dessas lojas.

Detalhe de Guarded Kettle Corporation (Classificação do cliente: 1)

Região	Produto	Data	Cliente	Quantidade	Receita	Custos de Vendas	Lucro
Oeste	A292	30/12/2008	Guarded Kettle Corporation	640	14891	7302	7589
Oeste	B722	30/12/2008	Guarded Kettle Corporation	210	4980	2615	2365
Oeste	E438	30/12/2008	Guarded Kettle Corporation	660	20360	10276	10084
Oeste	E438	30/12/2008	Guarded Kettle Corporation	590	18024	9186	8838
Oeste	D625	29/12/2008	Guarded Kettle Corporation	950	28677	13804	14873
Oeste	C409	02/01/2007	Guarded Kettle Corporation	770	20382	10387	9995
Oeste	E438	29/12/2008	Guarded Kettle Corporation	160	4552	2491	2061
Oeste	A292	29/12/2008	Guarded Kettle Corporation	750	17451	8558	8893
Oeste	A292	26/12/2008	Guarded Kettle Corporation	390	8561	4450	4111
Oeste	A292	24/12/2008	Guarded Kettle Corporation	970	22570	11068	11502
Oeste	C409	24/12/2008	Guarded Kettle Corporation	340	9529	4587	4942
Oeste	E438	19/12/2008	Guarded Kettle Corporation	800	22762	12456	10306

Listagem 13.10 Código utilizado para criar um relatório para cada um dos três maiores clientes

```
Sub RetrieveTop3CustomerDetail()
    ' Listagem 13.10
    ' Recupera detalhes das 3 maiores lojas
    Dim WSD As Worksheet
    Dim WSR As Worksheet
    Dim WBN As Workbook
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long
    Set WSD = Worksheets("PivotTable")

    ' Exclui todas as tabelas dinâmicas anteriores
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("J1:Z1").EntireColumn.Clear

    ' Define a área de entrada e configura um cache dinâmico
    FinalRow = WSD.Cells(Application.Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Application.Columns.Count).End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, SourceData:=PRange.Address)

    ' Cria a tabela dinâmica a partir do cache dinâmico
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD.Cells(2, FinalCol + 2), TableName:="TabelaDinâmica1")

    ' Desativa a atualização ao criar a tabela
    PT.ManualUpdate = True

    ' Configura os campos de linha
    PT.AddFields RowFields:="Cliente", ColumnFields:="Dados"

    ' Configura os campos de dados
    With PT.PivotFields("Receita")
```

```
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
        .NumberFormat = "#,##0"
        .Name = "Receita Total"
    End With

    ' Classifica as lojas em ordem decrescente por soma de receita
    PT.PivotFields("Cliente").AutoSort Order:=xlDescending,Field:="Receita Total"

    ' Mostra somente as 3 maiores lojas
    PT.PivotFields("Cliente").AutoShow Type:=xlAutomatic, Range:=xlTop,Count:=3, Field:="Receita Total"

    ' Assegura que obtemos zeros em vez de lacunas na área de dados
    PT.NullString = "0"

    ' Calcula a tabela dinâmica para permitir que o rótulo de data seja elaborado
    PT.ManualUpdate = False
    PT.ManualUpdate = True

    ' Produz relatórios de resumo para cada cliente
    For i = 1 To 3
        PT.TableRange2.Offset(i + 1, 1).Resize(1, 1).ShowDetail = True
        ' A planilha ativa mudou para o novo relatório de detalhe
        ' Adiciona um título
        Range("A1:A2").EntireRow.Insert
        Range("A1").Value = "Detalhe de " &PT.TableRange2.Offset(i + 1, 0).Resize(1, 1).Value &" _
            (Customer Rank: " & i & ")"
    Next i

    MsgBox "Relatórios detalhados dos 3 maiores clientes foram criados."
End Sub
```

Criando relatório para cada região ou modelo

Uma tabela dinâmica pode ter um ou mais campos Filtro de Relatório. Um campo Filtro de Relatório entra em um conjunto separado de linhas acima do relatório dinâmico. Ele serve para filtrar o relatório por uma certa região, por determinado modelo ou por uma combinação de região e modelo.

No VBA, os campos Filtro de Relatório são chamados *campos de página*.

Para configurar um campo de página no VBA, adicione o parâmetro `PageFields` ao método `AddFields`. A linha de código a seguir cria uma tabela dinâmica que coloca a região no campo de página:

```
PT.AddFields RowFields:= "Produto", ColumnFields:= "Dados", PageFields:= "Região"
```

A linha de código anterior configura o campo de página Região com o valor (All), que retorna todas regiões. Para limitar o relatório apenas à região Norte, utilize a propriedade `CurrentPage`:

```
PT.PivotFields("Região").CurrentPage = "Norte"
```

Um campo de página pode ser utilizado para criar um formulário para selecionar uma determinada região ou produto. Então, você utiliza essas informações para configurar a propriedade `CurrentPage` e exibir os resultados do formulário de usuário.

Outro uso interessante é fazer um loop por todos os `PivotItems` e exibi-los individualmente no campo de página. Você pode produzir rapidamente relatórios dos 10 maiores de cada região utilizando esse método.

Para determinar quantas regiões estarão disponíveis nos dados, use `PT.PivotFields("Região").PivotItems.Count`. Qualquer um destes loops funcionará:

```
For i = 1 To PT.PivotFields("Região").PivotItems.Count
    PT.PivotFields("Região").CurrentPage =PT.PivotFields("Região").PivotItems(i).Name
    PT.ManualUpdate = False
    PT.ManualUpdate = True
Next i

For Each PivItem In PT.PivotFields("Região").PivotItems
    PT.PivotFields("Região").CurrentPage = PivItem.Name
    PT.ManualUpdate = False
    PT.ManualUpdate = True

Next PivItem
```

Evidentemente, nesses dois loops, os três relatórios de região aparecem de maneira muito rápida para serem vistos. Na prática, seria recomendável salvar todos os relatórios durante sua exibição.

Até agora neste capítulo, você utilizou `PT.TableRange2` ao copiar os dados da tabela dinâmica. A propriedade `TableRange2` abrange todas linhas da tabela dinâmica, inclusive os campos de página. Também há uma propriedade `.TableRange1`, que exclui os campos de página. Você pode utilizar qualquer uma destas instruções para obter as linhas de detalhes:

```
PT.TableRange2.Offset(3, 0)

PT.TableRange1.Offset(1, 0)
```

Qual instrução utilizar depende de sua preferência; mas se utilizar `TableRange2`, você não irá ter problemas ao tentar excluir a tabela dinâmica com `PT.TableRange2.Clear`. Se, acidentalmente, tentar limpar `TableRange1` quando houver campos de página, você acabará vendo o temido erro "Você não pode mover nem pode alterar parte de uma tabela dinâmica".

A Listagem 13.11 produz uma nova pasta de trabalho para cada região. O relatório da região final é mostrado na Figura 13.28.

	A	B	C
1	Os 5 maiores clientes na Região Oeste		
2			
3	Cliente	Receita	
4	Guarded Kettle Corporation	8.889K	
5	Agile Glass Supply	3.877K	
6	Tremendous Flagpole Traders	2.361K	
7	Innovative Oven Corporation	2.359K	
8	Trouble-Free Eggbeater Inc.	2.303K	
9	Total dos 5 maiores	19.789K	

Figura 13.28
Fazendo loop por todos os itens localizados no campo de página Região, a macro produziu uma pasta de trabalho para cada gerente regional.

Listagem 13.11 Código que cria uma nova pasta de trabalho por região

```
Sub Top5ByRegionReport()
    ' Listagem 13.11
    ' Produz um relatório dos 5 maiores clientes para cada região
    Dim WSD As Worksheet
    Dim WSR As Worksheet
    Dim WBN As Workbook
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long

    Set WSD = Worksheets("TabelaDinamica")

    ' Exclui todas as tabelas dinâmicas anteriores
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("J1:Z1").EntireColumn.Clear

    ' Define a área de entrada e configura um cache dinâmico
    FinalRow = WSD.Cells(Application.Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Application.Columns.Count).End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, SourceData:=PRange.Address)

    ' Cria a tabela dinâmica a partir do cache dinâmico
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD.Cells(2, FinalCol + 2), TableName:="PivotTable1")

    ' Desativa a atualização ao criar a tabela
    PT.ManualUpdate = True

    ' Configura os campos de linha
    PT.AddFields RowFields:="Cliente", ColumnFields:="Dados",PageFields:="Região"

    ' Configura os campos de dados
    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
        .NumberFormat = "#,##0,K"
        .Name = "Receita Total"
```

```
    End With

    ' Classifica clientes em ordem decrescente por soma de receita
    PT.PivotFields("Cliente").AutoSort Order:=xlDescending,Field:="Receita Total"

    ' Mostra somente os 5 maiores clientes
    PT.PivotFields("Cliente").AutoShow Type:=xlAutomatic, Range:=xlTop,Count:=5, Field:="Receita Total"

    ' Assegura que obtemos zeros em vez de lacunas na área de dados
    PT.NullString = "0"

    ' Calcula a tabela dinâmica
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    Ctr = 0

    ' Faz loop por região
    For Each PivItem In PT.PivotFields("Região").PivotItems
        Ctr = Ctr + 1
        PT.PivotFields("Região").CurrentPage = PivItem.Name
        PT.ManualUpdate = False
        PT.ManualUpdate = True

        ' Cria uma nova pasta de trabalho em branco com uma planilha
        Set WBN = Workbooks.Add(xlWBATWorksheet)
        Set WSR = WBN.Worksheets(1)
        WSR.Name = PivItem.Name
        ' Configura o título do relatório
        With WSR.[A1]
            .Value = "5 maiores clientes na " & PivItem.Name & " Região"
            .Font.Size = 14
        End With

        ' Copie os dados da tabela dinâmica para a linha 3 da planilha de relatório
        ' Use deslocamento para eliminar as linhas de página & título da tabela dinâmica
        PT.TableRange2.Offset(3, 0).Copy
        WSR.[A3].PasteSpecial Paste:=xlPasteValuesAndNumberFormats
        LastRow = WSR.Cells(65536, 1).End(xlUp).Row
        WSR.Cells(LastRow, 1).Value = "Total dos 5 maiores"

        ' Aplica formatação básica
        ' Autoajusta colunas, aplica negrito aos títulos e os alinha à direita
        WSR.Range(WSR.Range("A2"), WSR.Cells(LastRow, 3)).Columns.AutoFit
        Range("A3").EntireRow.Font.Bold = True
        Range("A3").EntireRow.HorizontalAlignment = xlRight
        Range("A3").HorizontalAlignment = xlLeft
        Range("B3").Value = "Receita"

        Range("A2").Select

    Next PivItem

    ' Limpa a tabela dinâmica
    PT.TableRange2.Clear
    Set PTCache = Nothing

    MsgBox Ctr & " Os relatórios por Região foram criados"

End Sub
```

Filtrando manualmente dois ou mais itens em um campo dinâmico

Além de configurar um item dinâmico calculado para exibir o total de alguns produtos que compõem uma dimensão, você pode filtrar manualmente um determinado campo dinâmico.

Por exemplo, você tem um cliente que vende sapatos. No relatório de vendas de sandálias, ele quer ver apenas as lojas que estão nos estados de clima quente. O código para ocultar uma determinada loja é semelhante ao seguinte:

```
PT.PivotFields("Loja").PivotItems("Minneapolis").Visible = False
```

Você precisa ser muito cuidadoso para *nunca* definir todos os itens como `False`; caso contrário, a macro *termina em* erro. Isso tende a acontecer mais do que se imagina. Um aplicativo pode primeiro mostrar os produtos A e B e, depois, no próximo loop, mostrar os produtos C e D. Se você tentar ocultar A e B antes de tornar C e D visíveis, nenhum produto será visível no `PivotField`, o que causa erro. Para corrigir isso, sempre faça loop por todos os `PivotItems`, certificando-se de torná-los novamente visíveis antes da segunda passagem pelo loop.

Esse processo é fácil no VBA. Depois de criar a tabela com `Produto` no campo de página, faça loop alterando a propriedade `Visible` para mostrar apenas o total de certos produtos:

```
' Certifica-se de que todos os PivotItems na linha são visíveis
For Each PivItem InPT.PivotFields("Produto").PivotItems
    PivItem.Visible = True
Next PivItem

' Agora faz loop e mantém apenas certos itens visíveis
For Each PivItem InPT.PivotFields("Produto").PivotItems
    Select Case PivItem.Name
        Case "Landscaping/Grounds Care","Green Plants and Foliage Care"
            PivItem.Visible = True
        Case Else
            PivItem.Visible = False
    End Select
Next PivItem
```

Controlando a ordem de classificação manualmente

Se sua empresa sempre faz relatórios colocando as regiões na seqüência Leste, Central, Oeste, será uma batalha árdua fazer com que os gerentes aceitem o relatório na ordem Central, Leste e Oeste, porque essa é a ordem alfabética padrão oferecida por tabelas dinâmicas.

De modo bem estranho, a Microsoft oferece um método incomum para tratar uma ordem de classificação personalizada em uma tabela dinâmica. Ela é chamada *ordem de classificação manual*. Para alterar a ordem de classificação na interface com o usuário, você deve ir para a célula na tabela dinâmica que contém Central, digitar a palavra `Leste` e pressionar Enter. Como por mágica, Central e Leste mudam de lugar. Evidentemente, todos os números de Leste mudam para a coluna apropriada.

O código VBA para fazer uma classificação manual pede que se configure a propriedade `Position` para um `PivotItem` específico. Isso é um pouco perigoso, porque você não sabe se os campos subjacentes terão dados para Leste em algum determinado dia. Assegure-se de que a verificação de erros está configurada para retomar, caso Leste não exista na data atual:

```
On Error Resume Next
PT.PivotFields("Região").PivotItems("Leste").Position = 1
On Error GoTo 0
```

Utilizando Soma, Média, Contagem, Min, Max e Mais

Até agora, todos os exemplos neste capítulo envolveram soma de dados. Também é possível obter uma média, um mínimo ou um máximo dos dados. No VBA, mude a propriedade `Function` do campo de dados e dê um nome único para o campo de dados. Por exemplo, o fragmento de código a seguir produz cinco resumos diferentes do campo Receita, cada um com nome único:

```
'Configura os campos de dados
With PT.PivotFields("Receita")
    .Orientation = xlDataField
    .Function = xlSum
    .Position = 1
    .NumberFormat = "#,##0,K"
    .Name = "Receita Total"
End With

With PT.PivotFields("Receita")
    .Orientation = xlDataField
    .Function = xlCount
    .Position = 2
    .NumberFormat = "#,##0"
    .Name = "Pedido Nº"
End With

With PT.PivotFields("Receita")
    .Orientation = xlDataField
    .Function = xlAverage
```

```
        .Position = 3
        .NumberFormat = "#,##0"
        .Name = "Receita Média"
    End With

    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Function = xlMin
        .Position = 4
        .NumberFormat = "#,##0"
        .Name = "Menor Pedido"
    End With

    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Function = xlMax
        .Position = 5
        .NumberFormat = "#,##0"
        .Name = "Maior Pedido"
    End With
```

A tabela dinâmica resultante fornece várias estatísticas sobre a receita média, o maior pedido, o menor pedido e assim por diante, como mostrado na Figura 13.29.

Figura 13.29
Essa tabela dinâmica apresenta quatro visualizações de Soma de Receita. A coluna K é o cálculo normal. A coluna L é a percentagem do Total. A coluna M é o percentual de mudança do mês anterior. A coluna N é o total corrente.

J	K	L	M	N	O
	Dados				
Região	Receita Total	Número de Pedidos	Receita Média	Menor Pedido	Maior Pedido
Leste	24.331K	1.696	14.346	2K	32.023
Central	22.993K	1.625	14.149	2K	31.993
Oeste	23.769K	1.666	14.267	2K	32.320
Total geral	71.094K	4.987	14.256	2K	32.320

Criando as percentagens do relatório

Além das escolhas disponíveis, como Soma, Min, Max e Média, você pode utilizar outro conjunto de opções de tabela dinâmica chamado *opções de cálculo*. Essas opções permitem exibir um determinado campo como um percentagem do total, da linha e da coluna, ou como a diferença de percentagem entre o item anterior e o próximo. Todas essas configurações são controladas pela propriedade `.Calculation` do campo de página.

As propriedades válidas para `.Calculation` são `xlPercentOf`, `xlPercentOfColumn`, `xlPercentOfRow`, `xlPercentOfTotal`, `xlRunningTotal`, `xlPercentDifferenceFrom`, `xlDifferenceFrom`, `xlIndex` e `xlNoAdditionalCalculation`. Cada uma dessas propriedades tem seu próprio conjunto de regras. Algumas requerem que se especifique um `BaseField` e outras, um `BaseField` como um `BaseItem`. As seções a seguir fornecem alguns exemplos específicos.

Percentagem do total

Para obter a percentagem do total, especifique `xlPercentOfTotal` como a propriedade `.Calculation` para o campo de página:

```
    'Configura uma percentagem do total
    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Caption = "PorCentoDoTotal"
        .Function = xlSum
        .Position = 2
        .NumberFormat = "#0.0%"
        .Calculation = xlPercentOfTotal
    End With
```

Aumento da percentagem a partir do mês anterior

Com os meses de entrega sendo dispostos em colunas, talvez você queira ver a percentagem de crescimento de receita mês a mês. Pode-se configurar essa disposição com a configuração `xlPercentDifferenceFrom`. Nesse caso, você deve especificar que o `BaseField` é `"Data"` e que o `BaseItem` é algo chamado `(anterior)`:

```
' Configura % de alterações com base no mês anterior
With PT.PivotFields("Receita")
    .Orientation = xlDataField
    .Function = xlSum
    .Caption = "%Alterada"
    .Calculation = xlPercentDifferenceFrom
    .BaseField = "Data"
    .BaseItem = "(anterior)"
    .Position = 3
    .NumberFormat = "#0.0%"
End With
```

Observe que com cálculos posicionais não se pode utilizar o método `AutoShow` ou `AutoSort`. Isso é uma pena; seria interessante classificar os clientes de cima para baixo e ver o tamanho deles, um em relação ao outro.

Percentagem de um item específico

Você pode usar a configuração `xlPercentDifferenceFrom` como receitas expressas com uma percentagem das vendas da região Oeste:

```
'Mostra receita como uma percentagem da Califórnia
With PT.PivotFields("Receita")
    .Orientation = xlDataField
    .Function = xlSum
    .Caption = "% do Oeste"
    .Calculation = xlPercentDifferenceFrom
    .BaseField = "Região"
    .BaseItem = "Oeste"
    .Position = 3
    .NumberFormat = "#0.0%"
End With
```

Total corrente

Configurar um total corrente não é intuitivo; para fazer isso, você deve definir um `BaseField`. Nesse exemplo, Data executa para baixo pela coluna. Para definir uma coluna de total corrente para receita, você deve especificar que `BaseField` é `"Data"`:

```
'Configurando o total corrente
With PT.PivotFields("Receita")
    .Orientation = xlDataField
    .Function = xlSum
    .Caption = "Total corrente"
    .Calculation = xlRunningTotal
    .Position = 4
    .NumberFormat = "#,##0,K"
    .BaseField = "Data"
End With
```

A Figura 13.30 mostra os resultados de uma tabela dinâmica com três configurações de cálculo personalizadas, como discutido anteriormente.

Figura 13.30
Essa tabela dinâmica apresenta quatro visualizações de Soma de Receita. A coluna L é o cálculo normal. A coluna M é o percentagem do Total. A coluna N é a percentagem de mudança em relação ao mês anterior. A coluna O é o total corrente.

		Dados			
Anos	Data	Soma de Receita	PorCentoDoTotal	%Alterada	Total corrente
⊟2007	jan	2.610K	3,7%		2.610K
	fev	2.418K	3,4%	-7,3%	5.027K
	mar	2.642K	3,7%	9,3%	7.670K
	abr	3.104K	4,4%	17,5%	10.773K
	mai	3.233K	4,5%	4,2%	14.006K
	jun	2.642K	3,7%	-18,3%	16.648K
	jul	2.663K	3,7%	0,8%	19.311K
	ago	3.292K	4,6%	23,6%	22.603K
	set	3.010K	4,2%	-8,6%	25.613K
	out	2.687K	3,8%	-10,7%	28.300K
	nov	2.742K	3,9%	2,0%	31.042K
	dez	3.156K	4,4%	15,1%	34.197K
⊟2008	jan	3.097K	4,4%		3.097K
	fev	2.649K	3,7%	-14,4%	5.746K
	mar	2.866K	4,0%	8,2%	8.612K
	abr	2.895K	4,1%	1,0%	11.507K
	mai	3.231K	4,5%	11,6%	14.738K
	jun	2.768K	3,9%	-14,3%	17.506K
	jul	3.638K	5,1%	31,4%	21.144K
	ago	2.974K	4,2%	-18,2%	24.118K
	set	3.209K	4,5%	7,9%	27.327K
	out	3.485K	4,9%	8,6%	30.811K
	nov	2.740K	3,9%	-21,4%	33.551K
	dez	3.345K	4,7%	22,1%	36.896K
Total geral		71.094K	100,0%		

Utilizando os novos recursos da tabela dinâmica no Excel 2007

NOVO As tabelas dinâmicas oferecem uma variedade de novos recursos no Excel 2007. O novo rótulo e filtros de valor, formatação condicional, tabela formatar e visualizações de layout são aperfeiçoamentos significativos para o ambiente de tabela dinâmica.

Se você quiser utilizar qualquer um desses recursos, a tabela dinâmica deve existir em um arquivo armazenado no formato de arquivo do Excel 2007. Se o arquivo estiver em modo de compatibilidade, nenhum dos novos recursos estará disponível na interface com o usuário nem no VBA.

De maneira semelhante, se você utilizar qualquer um desses recursos, o código executará apenas no Excel 2007. Não há nenhuma esperança de compatibilidade para compartilhar o código com um usuário do Excel 2003.

Utilizando os novos filtros

Nas versões anteriores do Excel, o recurso da filtragem permitia escolher um ou mais itens dinâmicos de uma lista suspensa. O único filtro conceitual era o filtro AutoShow dos 10 maiores.

O Excel 2007 oferece novos filtros conceituais fáceis de serem acessados. Na Lista de Campos da Tabela Dinâmica, paire o cursor sobre qualquer campo ativo na parte da lista de campos da caixa de diálogo. Na lista suspensa que é mostrada, você pode escolher Filtros de Rótulo, Filtros de Data ou Filtros de Valor.

Na Figura 13.31, o menu flutuante mostra a lista de Filtros de Rótulo disponíveis para o campo Cliente.

Para aplicar um filtro de rótulo no VBA, utilize o método `PivotFilters.Add`. O código seguinte filtra os desvios que iniciam com 1:

```
PT.PivotFields("Cliente").PivotFilters.AddType:=xlCaptionBeginsWith, Value1:="1"
```

Figura 13.31
Você pode escolher facilmente todos os itens de Cliente que atendam a seus critérios.

Para limpar o filtro do campo Desvio, utilize o método `ClearAllFilters`:

```
PT.PivotFields("("Cliente").ClearAllFilters
```

Para aplicar um filtro de data ao campo Data para localizar registros dessa semana, utilize este código:

```
PT.PivotFields("Data").PivotFilters.Add Type:=xlThisWeek
```

Os filtros de valor permitem filtrar um campo baseado no valor de outro campo. Por exemplo, para localizar todos os desvios em que a receita total seja maior do que US$ 100 mil, utilize este código:

```
PT.PivotFields("Cliente").PivotFilters.AddType:=xlValueIsGreaterThan,DataField:=PT.PivotFields("Soma de
Receita"),Value1:=100000
```

Outros filtros de valor podem permitir que você especifique se quer desvios onde a receita esteja entre US$ 50 mil e US$ 100 mil. Nesse caso, você especifica o primeiro limite como `Value1` e o segundo como `Value2`:

```
PT.PivotFields("Desvio").PivotFilters.AddType:=xlValueIsBetween,DataField:=PT.PivotFields("Soma de
Receita"),Value1:=50000, Value2:=100000
```

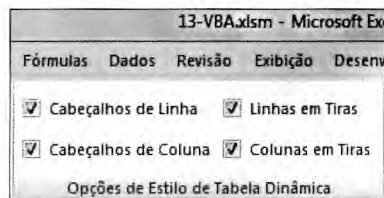
A Tabela 13.3 fornece uma amostragem de tipos de filtro.

Tabela 13.3 Amostragem de tipos de filtro

Tipo de filtro	Descrição
`xlBefore`	Filtra por todas as datas antes de uma data especificada
`xlAllDatesInPeriodJanuary`	Filtra por todas as datas de janeiro
`xlBottomCount`	Filtra pelo número especificado de valores da parte inferior de uma lista
`xlBottomPercent`	Filtra pela percentagem especificada de valores da parte inferior de uma lista
`xlBottomSum`	Soma os valores da parte inferior da lista
`xlCaptionBeginsWith`	Filtra por todas legendas que iniciam com a string especificada
`xlDateBetween`	Filtra por todas as datas que estão entre um intervalo especificado de datas
`xlDateLastMonth`	Filtra por todas as datas que se aplicam ao mês anterior
`xlValueEquals`	Filtra por todos os valores que correspondem ao valor especificado
`xlYearToDate`	Filtra por todos os valores que estão dentro um ano de uma data especificada

Aplicando um estilo de tabela

A guia Design oferece dois grupos dedicados à formatação da tabela dinâmica, como mostrado na Figura 13.32. O grupo Opções de Estilo de Tabela Dinâmica tem quatro caixas de seleção que modificam os estilos no grupo Estilos de Tabela Dinâmica.

Figura 13.32
As quatro caixas de seleção e o grupo de estilos oferecem muitas variações para formatar a tabela dinâmica.

As quatro linhas de código a seguir equivalem a ativar as quatro configurações no grupo Opções de Estilo de Tabela Dinâmica:

```
PT.ShowTableStyleRowHeaders = True
PT.ShowTableStyleColumnHeaders = True
PT.ShowTableStyleRowStripes = True
PT.ShowTableStyleColumnStripes = True
```

Aplicar um estilo de tabela da galeria utiliza a propriedade `TableStyle2`. Se você quiser obter o nome correto, talvez seja melhor gravar uma macro:

```
'Formata a tabela dinâmica
PT.ShowTableStyleRowStripes = True
PT.TableStyle2 = "PivotStyleMedium3"
```

Se você pairar o curso do mouse sobre um estilo no grupo Estilo, uma Dica de Tela mostrará um nome de estilo como Mídia Estilo Dinâmico 3.

Mudando o Layout na guia Design

O grupo Layout da guia (ou faixa) Design contém quatro listas suspensas que controlam a localização de subtotais (parte superior ou parte inferior), a presença de totais gerais, o layout de relatório e a presença de linhas em branco.

Os subtotais podem aparecer na parte superior ou inferior de um grupo de itens dinâmicos. A propriedade `SubtotalLocation` se aplica à tabela dinâmica inteira; os valores válidos são `xlAtBottom` ou `xlAtTop`:

```
PT.SubtotalLocation:=xlAtTop
```

Os totais gerais podem ser ativados ou desativados para linhas ou colunas. O código seguinte os desativa em ambos:

```
PT.ColumnGrand = False
PT.RowGrand = False
```

Há três configurações para o layout de relatório. O layout Tabular é semelhante ao layout padrão no Excel 2003. O layout Estrutura de Tópicos estava opcionalmente disponível no Excel 2003. O layout Compacto é novo no Excel 2007.

O Excel pode se lembrar do último layout utilizado e aplicá-lo a tabelas dinâmicas adicionais criadas na mesma sessão do Excel. Por essa razão, você deve sempre escolher explicitamente o layout que quer. Utilize o método `RowAxisLayout`; os valores válidos são `xlTabularRow`, `xlOutlineRow` ou `xlCompactRow`:

```
PT.RowAxisLayout xlTabularRow
PT.RowAxisLayout xlOutlineRow
PT.RowGrand = xlCompactRow
```

No Excel 2007, você pode adicionar uma linha em branco ao layout depois de cada grupo de itens dinâmicos. Embora a guia Design ofereça uma única configuração para afetar a tabela dinâmica inteira, a configuração é aplicada realmente a todos os campos dinâmicos individualmente. O programa de gravação de macros responde gravando uma dúzia de linhas de código para uma tabela dinâmica com 12 campos. É possível adicionar de maneira inteligente uma única linha de código ao(s) campo(s) de linha externo(s):

```
PT.PivotFields("Região").LayoutBlankLine = True
```

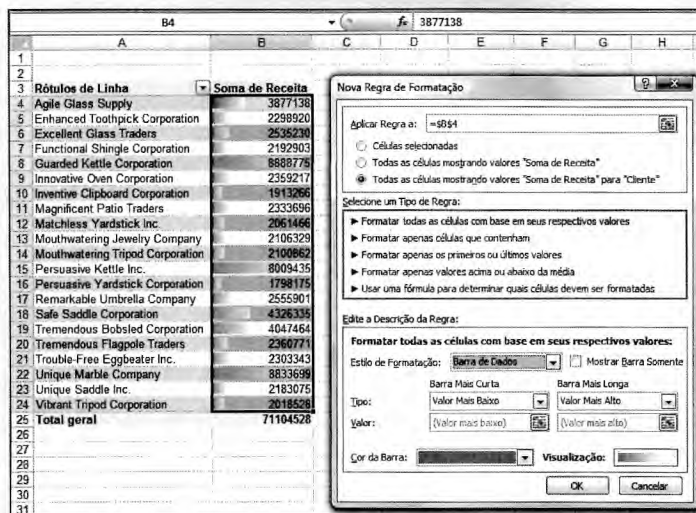
Estudo de caso

Aplicando uma visualização de dados

O Excel 2007 oferece novas e fantásticas visualizações de dados como conjunto de ícones, gradientes de cor e barras de dados na célula. Ao aplicar uma visualização a uma tabela dinâmica, você deve excluir as linhas de total da visualização.

Se tiver 20 clientes que apresentam a média de US$ 3 milhões em receita individualmente, o total para os 20 clientes será de US$ 60 milhões. Se você incluir o total na visualização de dados, o total ocupará a barra maior e todos os registros de clientes ficarão em barras muito pequenas.

Na interface com o usuário do Excel, você sempre selecionará Adicionar Regra ou Editar Regra para escolher a opção Todas as Células que mostram 'Soma de Receita' para 'Cliente', como mostrado na Figura 13.33.

Figura 13.33
Para criar visualizações significativas em sua tabela dinâmica, exclua os totais escolhendo a terceira opção na parte superior dessa caixa de diálogo.

O código na Listagem 13.12 adiciona uma tabela dinâmica e aplica uma barra de dados ao campo de receita.

Listagem 13.12 Código que cria uma tabela dinâmica com barras de dados

```
Sub Sub CreatePivotDataBar()
    ' Listagem 13.12
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long
    Set WSD = Worksheets("PivotTable")

    ' Exclui todas as tabelas dinâmicas anteriores
    For Each PT In WSD.PivotTables
```

```
        PT.TableRange2.Clear
    Next PT
    WSD.Range("J1:Z1").EntireColumn.Clear

    ' Define a área de entrada e configura um cache dinâmico
    FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Columns.Count).End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, SourceData:=PRange.Address)

    ' Cria a tabela dinâmica a partir do cache dinâmico
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD.Cells(2, FinalCol + 2), TableName:="PivotTable1")

    ' Desativa a atualização ao criar a tabela
    PT.ManualUpdate = True

    ' Configurando campos Linha & Coluna
    PT.AddFields RowFields:="Cliente",ColumnFields:="Dados"


    ' Configura os campos de dados
    With PT.PivotFields("Receita")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
    End With

    ' Calcula a tabela dinâmica
    PT.ManualUpdate = False
    PT.ManualUpdate = True

    ' Aplique uma barra de dados
    PT.TableRange2.Cells(3, 2).Select
    Selection.FormatConditions.AddDatabar
    Selection.FormatConditions(1).ShowValue = True
    Selection.FormatConditions(1).SetFirstPriority
    With Selection.FormatConditions(1)
        .MinPoint.Modify newtype:=xlConditionValueLowestValue
        .MaxPoint.Modify newtype:=xlConditionValueHighestValue
    End With
    With Selection.FormatConditions(1).BarColor
        .ThemeColor = xlThemeColorAccent3
        .TintAndShade = -0.499984740745262
    End With
    Selection.FormatConditions(1).ScopeType = xlFieldsScope

    WSD.Activate
    Range("J2").Select

End Sub
```

Próximos passos

Se você ainda não tinha certeza disso, saiba que as tabelas dinâmicas são meu recurso favorito no Excel. Elas são incrivelmente poderosas e flexíveis. Combinadas com o VBA, fornecem um excelente mecanismo de cálculo e aprimoram muitos dos relatórios que crio para meus clientes. No Capítulo 14, "O poder do Excel", você aprenderá diversas técnicas para tratar várias tarefas no VBA.

O poder do Excel

14

NESTE CAPÍTULO

Um grande segredo dos programadores bem-sucedidos é nunca perder tempo escrevendo o mesmo código duas vezes. Todos têm pequenos — ou grandes — fragmentos de código que são utilizados repetidamente. Outro grande segredo é nunca levar oito horas fazendo algo que pode ser feito em dez minutos — este livro é sobre isso!

Este capítulo contém programas cedidos por vários programadores de Excel muito competentes. Eles consideram tais programas úteis e esperam que possam ajudá-lo. Esses programas não apenas economizam tempo, mas também podem ensinar novas maneiras de resolver problemas comuns.

Esses programadores têm estilos de programação diferentes e nós não reescrevemos as contribuições deles. Ao revisar as linhas de código, você notará diferentes maneiras de fazer a mesma tarefa, como referenciar intervalos.

Operações de arquivo

Os utilitários a seguir lidam com o tratamento de arquivos em pastas. Ser capaz de fazer loop por uma lista de arquivos em uma pasta é uma tarefa útil.

Liste arquivos em um diretório

Contribuição cedida por Nathan P. Oliver, de Minneapolis, Minnesota. Nathan é consultor financeiro e desenvolvedor de aplicativos.

Esse programa retorna o nome de arquivo, o tamanho e a data modificada de todos os arquivos no diretório selecionado e suas subpastas:

```
Sub ExcelFileSearch()
Dim srchExt As Variant, srchDir As Variant, i As Long, j As Long
Dim strName As String, varArr(1 To 1048576, 1 To 3) As Variant
Dim strFileFullName As String
Dim ws As Worksheet
Dim fso As Object

Let srchExt = Application.InputBox("Insira extensão do arquivo",_
  "Solicitação de informações")
If srchExt = False And Not TypeName(srchExt) = "String" Then
    Exit Sub
End If

Let srchDir = BrowseForFolderShell
If srchDir = False And Not TypeName(srchDir) = "String" Then
    Exit Sub
End If

Application.ScreenUpdating = False

Set ws = ThisWorkbook.Worksheets.Add(Sheets(1))
On Error Resume Next
Application.DisplayAlerts = False
ThisWorkbook.Worksheets("FileSearch Results").Delete
Application.DisplayAlerts = True
On Error GoTo 0
ws.Name = "FileSearch Results"

Let strName = Dir$(srchDir & "\*" & srchExt)
```

```
Do While strName <> vbNullString
    Let i = i + 1
    Let strFileFullName = srchDir & strName
    Let varArr(i, 1) = strFileFullName
    Let varArr(i, 2) = FileLen(strFileFullName) \ 1024
    Let varArr(i, 3) = FileDateTime(strFileFullName)
    Let strName = Dir$()
Loop

Set fso = CreateObject("Scripting.FileSystemObject")
Call recurseSubFolders(fso.GetFolder(srchDir), varArr(), i, CStr(srchExt))
Set fso = Nothing

ThisWorkbook.Windows(1).DisplayHeadings = False
With ws
    If i > 0 Then
        .Range("A2").Resize(i, UBound(varArr, 2)).Value = varArr
        For j = 1 To i
            .Hyperlinks.Add anchor:=.Cells(j + 1, 1), Address:=varArr(j, 1)
        Next
    End If
    .Range(.Cells(1, 4), .Cells(1, .Columns.Count)).EntireColumn.Hidden = True
    .Range(.Cells(.Rows.Count, 1).End(xlUp)(2),.Cells(.Rows.Count, 1)).EntireRow.Hidden = True
    With .Range("A1:C1")
        .Value = Array("Nome completo", "Kilobytes", "Última modificação")
        .Font.Underline = xlUnderlineStyleSingle
        .EntireColumn.AutoFit
        .HorizontalAlignment = xlCenter
    End With
End With
Application.ScreenUpdating = True
End Sub

Private Sub recurseSubFolders(ByRef Folder As Object,ByRef varArr() _
    As Variant,ByRef i As Long,ByRef srchExt As String)
Dim SubFolder As Object
Dim strName As String, strFileFullName As String
For Each SubFolder In Folder.SubFolders
    Let strName = Dir$(SubFolder.Path & "\*" & srchExt)
    Do While strName <> vbNullString
        Let i = i + 1
        Let strFileFullName = SubFolder.Path & "\" & strName
        Let varArr(i, 1) = strFileFullName
        Let varArr(i, 2) = FileLen(strFileFullName) \ 1024
        Let varArr(i, 3) = FileDateTime(strFileFullName)
        Let strName = Dir$()
    Loop
    If i > 1048576 Then Exit Sub
    Call recurseSubFolders(SubFolder, varArr(), i, srchExt)
Next
End Sub

Private Function BrowseForFolderShell() As Variant
Dim objShell As Object, objFolder As Object
Set objShell = CreateObject("Shell.Application")
Set objFolder = objShell.BrowseForFolder(0, "Selecione uma pasta", 0, "C:\")
If Not objFolder Is Nothing Then
    On Error Resume Next
    If IsError(objFolder.Items.Item.Path) Then
        BrowseForFolderShell = CStr(objFolder)
    Else
        On Error GoTo 0
        If Len(objFolder.Items.Item.Path) > 3 Then
            BrowseForFolderShell = objFolder.Items.Item.Path &Application.PathSeparator
        Else
            BrowseForFolderShell = objFolder.Items.Item.Path
        End If
    End If
Else
    BrowseForFolderShell = False
End If
Set objFolder = Nothing: Set objShell = Nothing
End Function
```

Importe arquivos CSV

Contribuição cedida por Masaru Kaji, da cidade de Kobe, Japão. Masaru fornece consultoria do Excel no Colo's Excel Junk Room (www.puremis.net/excel/).

Se você se encontrar importando uma grande quantidade de arquivos de variáveis separadas por vírgulas (*comma-separated variable — CSV*) e, então, precisar voltar e excluí-los, esse programa foi feito para você. Esse programa abre rapidamente um CSV no Excel e exclui permanentemente o arquivo original:

```
Option Base 1

Sub OpenLargeCSVFast()
    Dim buf(1 To 16384) As Variant
    Dim i As Long
    'Altera a localização e nome de arquivo aqui
    Const strFilePath As String = "C:\temp\Test.CSV"

    Dim strRenamedPath As String
    strRenamedPath = Split(strFilePath, ".")(0) & "txt"

    With Application
        .ScreenUpdating = False
        .DisplayAlerts = False
    End With
    'Configura um array para FieldInfo abrir CSV
    For i = 1 To 16384
        buf(i) = Array(i, 2)
    Next
    Name strFilePath As strRenamedPath
    Workbooks.OpenText Filename:=strRenamedPath, DataType:=xlDelimited,Comma:=True, FieldInfo:=buf
    Erase buf
    ActiveSheet.UsedRange.Copy ThisWorkbook.Sheets(1).Range("A1")
    ActiveWorkbook.Close False
    Kill strRenamedPath
    With Application
        .ScreenUpdating = True
        .DisplayAlerts = True
    End With
End Sub
```

Leia o CSV inteiro para a memória e faça a análise sintática

Contribuição cedida por Suat Mehmet Ozgur, de Istambul, Turquia. Suat desenvolve aplicativos em Excel, Access e Visual Basic para MrExcel.com e TheOfficeExperts.com.

Esse exemplo adota uma abordagem diferente para ler um arquivo de texto. Em vez de ler um registro por vez, a macro carrega o arquivo de texto inteiro na memória em uma única variável alfanumérica e, então, analisa sintaticamente a string em registros individuais. A vantagem desse método é que você acessa o arquivo em disco apenas uma vez. Todo processamento subseqüente ocorre na memória e é muito rápido:

```
Sub ReadTxtLines()
'Não é necessário instalar a biblioteca Scripting Runtime, porque usamos a vinculação tardia
Dim sht As Worksheet
Dim fso As Object
Dim fil As Object
Dim txt As Object
Dim strtxt As String
Dim tmpLoc As Long

    'Trabalhando na planilha ativa
    Set sht = ActiveSheet
    'Limpa dados na planilha
   sht.UsedRange.ClearContents

    'O objeto de sistema de arquivos de que precisamos para gerenciar arquivos
    Set fso = CreateObject("Scripting.FileSystemObject")

    'O arquivo que queremos abrir e ler
    Set fil = fso.GetFile("c:\test.txt")
```

```
    'Abrindo arquivo como um TextStream
    Set txt = fil.OpenAsTextStream(1)

    'Lendo o arquivo incluso em uma variável alfanumérica de uma vez
    strtxt = txt.ReadAll

    'Fecha textstream e libera o arquivo. Não precisamos mais dele.
    txt.Close

    'Localiza a primeira colocação do caractere nova linha
    tmpLoc = InStr(1, strtxt, vbCrLf)

    'Faz loop até não encontrar mais caracteres de nova linha
    Do Until tmpLoc = 0
        'Utiliza a coluna A e a próxima célula vazia para gravar a linha de arquivo de texto
        s ht.Cells(sht.Rows.Count, 1).End(xlUp).Offset(1).Value =Left(strtxt, tmpLoc - 1)

        'Remove a linha analisada sintaticamente a partir da variável em que armazenamos o arquivo incluso
        strtxt = Right(strtxt, Len(strtxt) - tmpLoc - 1)

        'Localiza a próxima posição do caractere de nova linha
        tmpLoc = InStr(1, strtxt, vbCrLf)
    Loop

    ' Última linha que tem dados, mas nenhum caractere de nova linha
    sht.Cells(sht.Rows.Count, 1).End(xlUp).Offset(1).Value = strtxt

    ' Já será liberado no fim desse procedimento mas,
    ' como bom hábito, configura o objeto como nada.
    Set fso = Nothing
End Sub
```

Combinando e separando pastas de trabalho

Os próximos quatro utilitários demonstram como combinar planilhas em pastas de trabalho únicas ou separar uma pasta de trabalho única em planilhas individuais ou documento do Word.

Separe planilhas em pastas de trabalho

Contribuição cedida por Tommy Miles, de Houston, Texas.

Esse exemplo passa pela pasta de trabalho ativa e salva cada planilha como sua própria pasta de trabalho no mesmo caminho como a pasta de trabalho original. O exemplo nomeia as novas pastas de trabalho baseando-se no nome da planilha. Isso sobrescreverá arquivos sem solicitação.

Você também notará que precisa escolher se salva o arquivo como xlsm (habilitado para macro) ou xlsx (as macros estarão *stripped*). Inclui ambas as linhas xlsm ou xlsx no código seguinte, mas removi o comentário das linhas xlsx, tornando-as inativas:

```
Sub SplitWorkbook()

Dim ws As Worksheet
Dim DisplayStatusBar As Boolean

DisplayStatusBar = Application.DisplayStatusBar
Application.DisplayStatusBar = True
Application.ScreenUpdating = False
Application.DisplayAlerts = False

For Each ws In ThisWorkbook.Sheets
    Dim NewFileName As String
    Application.StatusBar = ThisWorkbook.Sheets.Count & " Remaining Sheets"
    If ThisWorkbook.Sheets.Count <> 1 Then
        NewFileName = ThisWorkbook.Path & "\" & ws.Name & ".xlsm" 'Capacitado para macro
'       NewFileName = ThisWorkbook.Path & "\" & ws.Name & ".xlsx" _ 'Não capacitado para macro
        ws.Copy
        ActiveWorkbook.Sheets(1).Name = "Sheet1"
        ActiveWorkbook.SaveAs Filename:=NewFileName,FileFormat:=xlOpenXMLWorkbookMacroEnabled
'       ActiveWorkbook.SaveAs Filename:=NewFileName,FileFormat:=xlOpenXMLWorkbook
        ActiveWorkbook.Close SaveChanges:=False
    Else
```

```
        NewFileName = ThisWorkbook.Path & "\" & ws.Name & ".xlsm"
'       NewFileName = ThisWorkbook.Path & "\" & ws.Name & ".xlsx"
        ws.Name = "Sheet1"
    End If
Next

Application.DisplayAlerts = True
Application.StatusBar = False
Application.DisplayStatusBar = DisplayStatusBar
Application.ScreenUpdating = True
End Sub
```

Combine pastas de trabalho

Contribuição cedida por Tommy Miles.

Esse exemplo passa por todos os arquivos do Excel em um diretório especificado, combina-os em uma única pasta de trabalho e renomeia as planilhas com base no nome da pasta de trabalho original:

```
Sub CombineWorkbooks()
    Dim CurFile As String, DirLoc As String
    Dim DestWB As Workbook
    Dim ws As Object 'permite tipos diferentes de planilha

    DirLoc = ThisWorkbook.Path & "\tst\" 'local dos arquivos
    CurFile = Dir(DirLoc & "*.xls")

    Application.ScreenUpdating = False
    Application.EnableEvents = False

    Set DestWB = Workbooks.Add(xlWorksheet)

    Do While CurFile <> vbNullString
        Dim OrigWB As Workbook
        Set OrigWB = Workbooks.Open(Filename:=DirLoc & CurFile, ReadOnly:=True)

        ' Limita a nomes de planilha válidos e remove .xls*
        CurFile = Left(Left(CurFile, Len(CurFile) - 5), 29)

        For Each ws In OrigWB.Sheets
            ws.Copy After:=DestWB.Sheets(DestWB.Sheets.Count)

            If OrigWB.Sheets.Count > 1 Then
                DestWB.Sheets(DestWB.Sheets.Count).Name = CurFile & ws.Index
            Else
                DestWB.Sheets(DestWB.Sheets.Count).Name = CurFile
            End If
        Next

        OrigWB.Close SaveChanges:=False
        CurFile = Dir
    Loop

    Application.DisplayAlerts = False
        DestWB.Sheets(1).Delete
    Application.DisplayAlerts = True

    Application.ScreenUpdating = True
    Application.EnableEvents = True

    Set DestWB = Nothing
End Sub
```

Filtre e copie dados para planilhas separadas

Contribuição cedida por Dennis Wallentin, de Ostersund, Suécia. Dennis fornece dicas e truques sobre o Excel em www.xldennis.com.

Esse exemplo usa uma coluna especificada para filtrar dados e copia os resultados para novas planilhas na pasta de trabalho ativa.

```
Sub Filter_NewSheet()
Dim wbBook As Workbook
Dim wsSheet As Worksheet
```

```
Dim rnStart As Range, rnData As Range
Dim i As Long

Set wbBook = ThisWorkbook
Set wsSheet = wbBook.Worksheets("Sheet1")

With wsSheet
    'Certifica-se de que a primeira linha contém títulos.
    Set rnStart = .Range("A2")
    Set rnData = .Range(.Range("A2"), .Cells(.Rows.Count, 3).End(xlUp))
End With

Application.ScreenUpdating = True

For i = 1 To 5
    'Aqui filtramos os dados com o primeiro critério.
    rnStart.AutoFilter Field:=1, Criteria1:="AA" & i
    'Copia a lista filtrada.
    rnData.SpecialCells(xlCellTypeVisible).Copy
    'Adiciona uma nova planilha à pasta de trabalho ativa.
    Worksheets.Add Before:=wsSheet
    'Nomeia as novas planilhas adicionadas.
    ActiveSheet.Name = "AA" & i
    'Cola a lista de filtros.
    Range("A2").PasteSpecial xlPasteValues
Next i

'Redefine a lista ao seu status original.
rnStart.AutoFilter Field:=1

With Application
    'Redefine a área de transferência.
    .CutCopyMode = False
    .ScreenUpdating = False
End With

End Sub
```

Exporte dados para o Word

Contribuição cedida por Dennis Wallentin.

Esse programa transfere os dados do Excel para a primeira tabela localizada em um documento do Word. Ele utiliza a vinculação inicial; portanto, uma referência deve ser estabelecida no Editor do VB (utilizando Ferramentas, Referências) para a Biblioteca de Objetos do Microsoft Word:

```
Sub Export_Data_Word_Table()
Dim wdApp As Word.Application
Dim wdDoc As Word.Document
Dim wdCell As Word.Cell
Dim i As Long
Dim wbBook As Workbook
Dim wsSheet As Worksheet
Dim rnData As Range
Dim vaData As Variant

Set wbBook = ThisWorkbook
Set wsSheet = wbBook.Worksheets("Sheet1")

With wsSheet
    Set rnData = .Range("A1:A10")
End With

'Adiciona os valores no intervalo a um array de variantes dimensionais.
vaData = rnData.Value

'Aqui instanciamos o novo objeto.
Set wdApp = New Word.Application
'Aqui o documento-alvo reside na mesma pasta que a pasta de trabalho.
Set wdDoc = wdApp.Documents.Open(ThisWorkbook.Path & "\Test.docx")
```

```
'Importa dados para a primeira tabela e na primeira coluna de uma tabela de dez linhas.
For Each wdCell In wdDoc.Tables(1).Columns(1).Cells
    i = i + 1
    wdCell.Range.Text = vaData(i, 1)
Next wdCell

'Salva e fecha o documento.
With wdDoc
    .Save
    .Close
End With

'Fecha a instância oculta do Microsoft Word.
wdApp.Quit
'Libera as variáveis externas da memória.
Set wdDoc = Nothing
Set wdApp = Nothing

MsgBox "Os dados foram transferidos para Test.docx.", vbInformation

End Sub
```

Trabalhando com comentários de célula

Comentários de célula são recursos muito subutilizados do Excel. Os quatro utilitários seguintes ajudam a tirar o máximo proveito dos comentários de célula.

Liste comentários

Contribuição cedida por Tommy Miles.

O Excel permite que o usuário imprima os comentários em uma pasta de trabalho, mas não especifica a pasta de trabalho ou a planilha em que os comentários aparecem, apenas a célula, como mostrado na Figura 14.1. O próximo exemplo coloca os comentários, o autor e o local de cada comentário em uma nova planilha para que possam ser visualizados, salvos ou impressos mais facilmente. A Figura 14.2 mostra resultados de exemplo.

Célula: C5
Comentário: Bil Jelen:
Não inclui venda especial.

Célula: D14
Comentário: Bil Jelen:
Obrigado por baixar nossos arquivos de projetos.

Célula: D14
Comentário: Bil Jelen:
Visite MrExcel.com para mais de 70.000 artigos sobre o Microsoft Excel.

Figura 14.1
O Excel imprime apenas o endereço de célula de origem e seu comentário.

	A	B	C	D	E	F	G	H	I	J
1	Autor	Livro	Planilha	Intervalo	Comentário					
2	Bill Jelen	ListComments.xlsm	Plan1	$C$5	Não inclui venda especial.					
3	Bill Jelen	ListComments.xlsm	Plan1	$D$14	Obrigado por baixar nossos arquivos de projetos.					
4	Bill Jelen	ListComments.xlsm	Plan1	$A$27	Visite MrExcel.com para mais de 70.000 artigos sobre o Microsoft Excel.					
5										

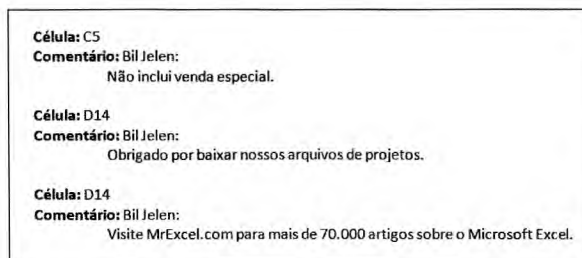



Figura 14.2
Liste facilmente todas as informações que pertencem aos comentários.

```
Sub ListComments()
    Dim wb As Workbook
    Dim ws As Worksheet

    Dim cmt As Comment

    Dim cmtCount As Long

    cmtCount = 2

    On Error Resume Next
        Set ws = ActiveSheet
            If ws Is Nothing Then Exit Sub
```

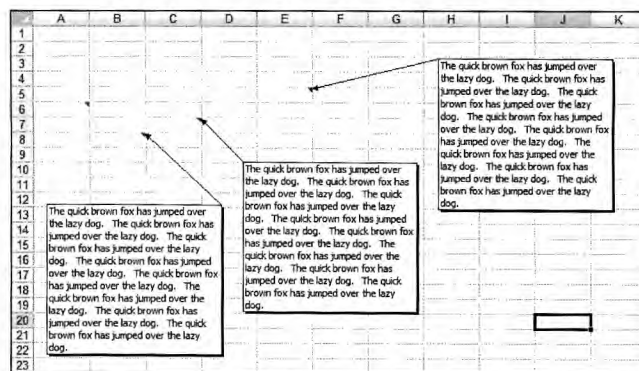
```
    On Error GoTo 0

    Application.ScreenUpdating = False

    Set wb = Workbooks.Add(xlWorksheet)

    With wb.Sheets(1)
        .Range("$A$1") = "Autor"
        .Range("$B$1") = "Livro"
        .Range("$C$1") = "Planilha"
        .Range("$D$1") = "Intervalo"
        .Range("$E$1") = "Comentário"
    End With

    For Each cmt In ws.Comments
        With wb.Sheets(1)
            .Cells(cmtCount, 1) = cmt.author
            .Cells(cmtCount, 2) = cmt.Parent.Parent.Parent.Name
            .Cells(cmtCount, 3) = cmt.Parent.Parent.Name
            .Cells(cmtCount, 4) = cmt.Parent.Address
            .Cells(cmtCount, 5) = CleanComment(cmt.author, cmt.Text)
        End With

        cmtCount = cmtCount + 1
    Next

    wb.Sheets(1).UsedRange.WrapText = False

    Application.ScreenUpdating = True

    Set ws = Nothing
    Set wb = Nothing
End Sub

Private Function CleanComment(author As String, cmt As String) As String
    Dim tmp As String

    tmp = Application.WorksheetFunction.Substitute(cmt, author & ":", "")
    tmp = Application.WorksheetFunction.Substitute(tmp, Chr(10), "")

    CleanComment = tmp
End Function
```

Redimensione comentários

Contribuição cedida por Tom Urtis, de São Francisco, Califórnia. Tom é o principal proprietário da Atlas Programming Management, uma empresa de consultoria em Excel na área da baía de São Francisco.

O Excel não redimensiona automaticamente os comentários de célula. E se houver vários em uma planilha, como mostrado na Figura 14.3, pode ser uma trabalheira redimensioná-los um por vez. O código de exemplo a seguir redimensiona todas as caixas de comentários em uma planilha para que, quando selecionadas, o comentário inteiro seja facilmente visualizável, como mostrado na Figura 14.4.

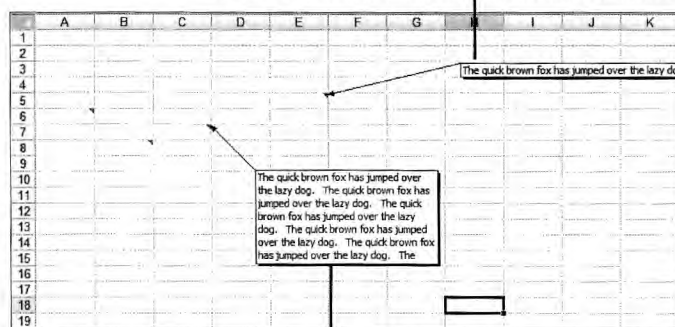
Figura 14.3
Por padrão, o Excel não dimensiona as caixas de comentários para mostrar todo o texto inserido.

Figura 14.4
Redimensione as caixas de comentários para ajustar todo o texto.

```
Sub CommentFitter1()
Application.ScreenUpdating = False
Dim x As Range, y As Long

For Each x In Cells.SpecialCells(xlCellTypeComments)
    Select Case True
        Case Len(x.NoteText) <> 0
            With x.Comment
                .Shape.TextFrame.AutoSize = True
                If .Shape.Width > 250 Then
                    y = .Shape.Width * .Shape.Height
                    .Shape.Width = 150
                    .Shape.Height = (y / 200) * 1.3
                End If
            End With
    End Select
Next x
Application.ScreenUpdating = True
End Sub
```

Redimensione comentários centralizando

Contribuição cedida por Tom Urtis.

Esse exemplo redimensiona todas as caixas de comentários em uma planilha, centralizando os comentários (veja Figura 14.5).

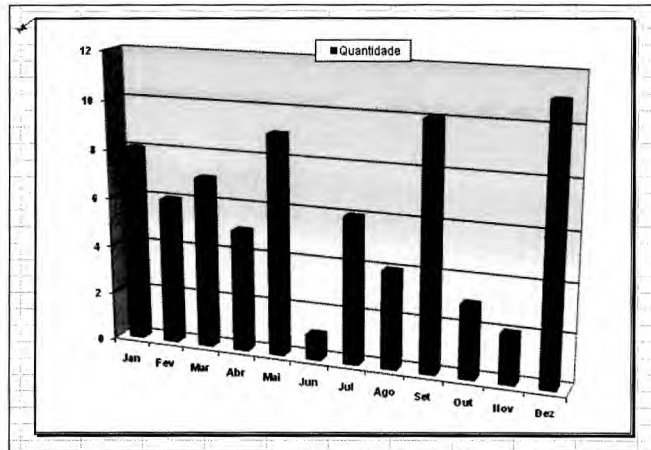
Figura 14.5
Centralize todos os comentários em uma planilha.

```
Sub CommentFitter2()
Application.ScreenUpdating = False
Dim x As Range, y As Long

For Each x In Cells.SpecialCells(xlCellTypeComments)
    Select Case True
        Case Len(x.NoteText) <> 0
            With x.Comment
                .Shape.TextFrame.AutoSize = True
                If .Shape.Width > 250 Then
                    y = .Shape.Width * .Shape.Height
                    .Shape.ScaleHeight 0.9, msoFalse, msoScaleFromTopLeft
                    .Shape.ScaleWidth 1#, msoFalse, msoScaleFromTopLeft
```

```
            End If
        End With
    End Select
Next x
Application.ScreenUpdating = True
End Sub
```

Coloque um gráfico em um comentário

Contribuição cedida por Tom Urtis.

Um gráfico ativo não pode existir em uma forma, mas você pode carregar a imagem do gráfico na forma de comentário, como mostrado na Figura 14.6.

Figura 14.6
Coloque um gráfico em um comentário de célula.

Eis os passos para fazer isso manualmente:

1. Crie e salve a imagem da figura que você quer que o comentário exiba.
2. Crie o comentário, se já não tiver feito isso, e selecione a célula em que ele se encontra.
3. Na guia Revisão, escolha Editar Comentário ou clique com o botão direito do mouse na célula e selecione Editar Comentário.
4. Clique com o botão direito do mouse na borda de comentário e selecione Formatar Comentário.
5. Selecione as guias Cores e Linhas e clique na seta para baixo, que pertence ao campo Cor da seção Preenchimento.
6. Selecione Efeitos de Preenchimento, selecione a guia Figura e então clique no botão Selecionar Figura.
7. Navegue para a imagem desejada, selecione-a e clique em OK duas vezes.

O efeito de ter um 'gráfico ativo' em um comentário pode ser alcançado se, por exemplo, o código fizer parte de um evento `SheetChange` quando os dados originais do gráfico estiverem sendo alterados. Além disso, os gráficos de negócios são muito atualizados; então, talvez você queira uma macro que mantenha o comentário atualizado e evite repetir os mesmos passos. A macro a seguir faz exatamente isto: modifica a macro para o nome de caminho de arquivo, o nome de gráfico, a planilha de destino e a célula e o tamanho de forma do comentário, dependendo do tamanho do gráfico.

```
Sub PlaceGraph()
Dim x As String, z As Range

Application.ScreenUpdating = False

'Atribui uma localização temporária para armazenar a imagem.
x = "C:\XWMJGraph.gif"

'Atribui a célula para armazenar o comentário.
Set z = Worksheets("ChartInComment").Range("A3")

'Exclui qualquer comentário existente na célula.
On Error Resume Next
z.Comment.Delete
On Error GoTo 0
```

```
'Seleciona e exporta o gráfico.
ActiveSheet.ChartObjects("Chart 1").Activate
ActiveChart.Export x

'Adiciona um novo comentário à célula, configura o tamanho e insere o gráfico.
With z.AddComment
    With .Shape
        .Height = 322
        .Width = 465
        .Fill.UserPicture x
    End With
End With

'Exclui a imagem temporária.
Kill x

Range("A1").Activate
Application.ScreenUpdating = True

Set z = Nothing
End Sub
```

Utilitários para impressionar seus clientes

Os quatro próximos utilitários surpreenderão e impressionarão seus clientes.

Utilizando a formatação condicional para destacar a célula selecionada

Contribuição cedida por Ivan F. Moala, de Auckland, Nova Zelândia. Ivan é o autor do site Web XcelFiles (www.xcelfiles.com), onde você descobrirá como fazer coisas que achava que não eram possíveis no Excel.

A formatação condicional é utilizada para destacar a linha e a coluna da célula ativa para ajudá-lo a localizá-la visualmente, como mostrado na Figura 14.7. Importante: Não utilize esse método se a planilha já tiver formatos condicionais, pois todos serão sobrescritos. Além disso, esse programa limpa a área de transferência, de modo que não é possível utilizá-lo ao copiar, recortar ou colar.

	A	B	C	D
4	Pedro	Leste	T1	Calça
5	Mário	Oeste	T2	Camisa
6	Luiz	Sul	T3	Sapato
7	Fernanda	Norte	T4	Jaqueta
8	Denise	Sul	T2	Camisa
9	Lúcio	Oeste	T4	Calça
10	Edson	Norte	T1	Jaqueta
11	João	Leste	T4	Jaqueta
12	Renata	Oeste	T2	Sapato
13	Valdir	Norte	T4	Camisa
14	Arnaldo	Sul	T1	Sapato
15	Cristina	Leste	T3	Jaqueta
16	Denise	Norte	T1	Calça
17	Mário	Oeste	T4	Camisa
18	Fernanda	Oeste	T3	Jaqueta
19	Sandra	Sul	T2	Calça
20	Gustavo	Leste	T1	Sapato
21	Antônio	Norte	T1	Sapato
22	Jeremias	Oeste	T2	Calça
23	Sebastião	Sul	T4	Jaqueta

Figura 14.7
Use a formatação condicional para destacar a célula selecionada em uma tabela.

```
Const iInternational As Integer = Not (0)

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
Dim iColor As Integer
'// Em caso de erro, retoma como caso.
'// Usuário seleciona um intervalo de células.
On Error Resume Next
iColor = Target.Interior.ColorIndex
'// Manter On Error ON em erros de deslocamento de linhas.

If iColor < 0 Then
    iColor = 36
Else
    iColor = iColor + 1
End If

'// Precisa desse teste caso a cor da fonte seja a mesma.
If iColor = Target.Font.ColorIndex Then iColor = iColor + 1
```

```
Cells.FormatConditions.Delete

'// Cor horizontal em faixas.
With Range("A" & Target.Row, Target.Address) 'Rows(Target.Row)
    .FormatConditions.Add Type:=2, Formula1:=iInternational 'Or just 1 '"TRUE"
    .FormatConditions(1).Interior.ColorIndex = iColor
End With

'// Cor vertical em faixas.
With Range(Target.Offset(1 - Target.Row, 0).Address & ":" &Target.Offset(-1, 0).Address)
    .FormatConditions.Add Type:=2, Formula1:=iInternational 'Or just 1 '"TRUE"
    .FormatConditions(1).Interior.ColorIndex = iColor
End With

End Sub
```

Realce a célula selecionada sem usar a formatação condicional

Contribuição cedida por Ivan F. Moala.

Esse exemplo destaca visualmente a célula ativa sem utilizar a formatação condicional quando as teclas de seta do teclado são utilizadas para mover-se pela planilha.

Coloque o seguinte código em um módulo padrão:

```
Dim strCol As String
Dim iCol As Integer
Dim dblRow As Double

Sub HighlightRight()
    HighLight 0, 1
End Sub

Sub HighlightLeft()
    HighLight 0, -1
End Sub

Sub HighlightUp()
    HighLight -1, 0, -1
End Sub

Sub HighlightDown()
    HighLight 1, 0, 1
End Sub

Sub HighLight(dblxRow As Double, iyCol As Integer, Optional dblZ As Double = 0)

On Error GoTo NoGo
strCol = Mid(ActiveCell.Offset(dblxRow, iyCol).Address,InStr(ActiveCell.Offset(dblxRow, iyCol).Address, "$") +
1,InStr(2, ActiveCell.Offset(dblxRow, iyCol).Address, "$") - 2)
iCol = ActiveCell.Column
dblRow = ActiveCell.Row

Application.ScreenUpdating = False

With Range(strCol & ":" & strCol & "," & dblRow + dblZ & ":" & dblRow + dblZ)
    .Select
    Application.ScreenUpdating = True
    .Item(dblRow + dblxRow).Activate
End With

NoGo:
End Sub

Sub ReSet() 'reinicialização manual
    Application.OnKey "{RIGHT}"
    Application.OnKey "{LEFT}"
    Application.OnKey "{UP}"
    Application.OnKey "{DOWN}"
End Sub
```

Coloque o seguinte código no módulo ThisWorkbook:

```
Private Sub Workbook_Open()
    Application.OnKey "{RIGHT}", "HighlightRight"
    Application.OnKey "{LEFT}", "HighlightLeft"
    Application.OnKey "{UP}", "HighlightUp"
    Application.OnKey "{DOWN}", "HighlightDown"
    Application.OnKey "{DEL}", "DisableDelete"
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Application.OnKey "{RIGHT}"
    Application.OnKey "{LEFT}"
    Application.OnKey "{UP}"
    Application.OnKey "{DOWN}"
    Application.OnKey "{DEL}"
End Sub
```

Transposição de dados personalizada

Contribuição cedida por Masaru Kaji.

Você tem um relatório no qual os dados são configurados em linhas (veja Figura 14.8), mas você precisa que eles sejam formatados para que todas as datas e lotes estejam em uma única linha, acompanhados pelo Valor e Posição Final (a Posição Final não é mostrada na Figura 14.9). O programa a seguir faz uma transposição de dados personalizada com base na coluna especificada, como mostrado na Figura 14.9.

Figura 14.8
Os dados originais têm registros semelhantes em linhas separadas.

	A	B	C	D	E
1	NomeDoItem	DataDoItem	NúmeroDoLote	PosiçãoFinal	Valor
2	Thermal	23/10/2002	1	8	2,15
3	Thermal	23/10/2002	1	3	3,2
4	Thermal	23/10/2002	1	2	4,9
5	Thermal	23/10/2002	1	1	6,1
6	Thermal	23/10/2002	1	7	6,2
7	Thermal	23/10/2002	1	4	12,9
8	Thermal	23/10/2002	1	9	23
9	Thermal	23/10/2002	1	5	36
10	Thermal	23/10/2002	1	6	36,25
11	Thermal	23/10/2002	2	2	1,05
12	Thermal	23/10/2002	2	1	2,5
13	Thermal	23/10/2002	2	8	7,3
14	Thermal	23/10/2002	2	3	10,9
15	Thermal	23/10/2002	2	4	12,1
16	Thermal	23/10/2002	2	9	21,7
17	Thermal	23/10/2002	2	6	33,25

Figura 14.9
O dados formatados transpõem os dados para que datas e lotes idênticos sejam mesclados em uma única linha.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	NomeDoItem	DataDoItem	NúmeroDoLote	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12
2	Thermal	23/10/2002	1	2,15	3,2	4,9	6,1	6,2	12,9	23	36	36,25			
3	Thermal	23/10/2002	2	1,05	2,5	7,3	10,9	12,1	21,7	33,25	43	43,25			
4	Thermal	23/10/2002	3	1,65	3,1	3,1	3,75	7,1	7,1	7,7	18,7	34	55,5		
5	Thermal	23/10/2002	4	1,1	2,75	4	9,5	14,3	25	37,75					
6	Thermal	23/10/2002	5	0,9	3,75	7,1	9	16	18,1	19,5	22,5	74,75			
7	Thermal	23/10/2002	6	1,6	3,4	5,2	7,8	8,2	9,4	11,5					
8	Thermal	23/10/2002	7	0,8	4,2	4,9	9,6	15	21,2	24,75	63,25				
9	Thermal	23/10/2002	8	0,7	6,2	8,4	10,3	10,6	12,3	28,75	31,75	52	76,75		
10	Thermal	23/10/2002	9	2,9	3,9	4,4	5,9	7	11,4	13,5	18,4	26,25	66,25		
11	Thermal	24/10/2002	1	1,4	3,85	6,2	8,1	10	12,3	17,2	27,5	37,5	55,5		
12	Thermal	24/10/2002	2	1,75	2,95	6	6,5	7,8	8,3	16,8					
13	Thermal	24/10/2002	3	1,15	5,4	8,7	9,9	10,9	11,8	13,3	17,1	24	37		
14	Thermal	24/10/2002	4	1,05	1,9	5,2	6,8	19,9							
15	Thermal	24/10/2002	5	2,5	3,15	3,15	4,2	6	12,2	12,3	19,9	23,2	25,25	42,25	150
16	Thermal	24/10/2002	6	2,4	2,95	4,4	6,5	8,7	14,2	22,9	22,9	25,75	51	58,25	59,5
17	Thermal	24/10/2002	7	1,2	3,35	6,3	9,5	11,3	12	14,4	36,25				
18	Thermal	24/10/2002	8	0,85	5	6,5	6,8	11,1	11,4	22,6					
19	Thermal	24/10/2002	9	2	3,3	4,5	6,8	8,6	9,7	20,5	30	58,5			
20	Thermal	25/10/2002	1	2,05	2,65	3,95	4,8	4,8	15,5	21	30	31,5	64,75	107,25	
21	Thermal	25/10/2002	2	1,25	2,25	7,5	9,1	9,1	12,3	27,25					
22	Thermal	25/10/2002	3	1	1	3,6	7,5	7,5	11,5	12,1	14,8	15,4	17,7		

```
Sub TransposeData()
Dim shOrg As Worksheet, shRes As Worksheet
Dim rngStart As Range, rngPaste As Range
Dim lngData As Long

Application.ScreenUpdating = False
On Error Resume Next
Application.DisplayAlerts = False
Sheets("TransposeResult").Delete
Application.DisplayAlerts = True
On Error GoTo 0

On Error GoTo terminate

Set shOrg = Sheets("TransposeData")
```

```
Set shRes = Sheets.Add(After:=shOrg)
shRes.Name = "TransposeResult"
With shOrg
    '--Classifica.
    .Cells.CurrentRegion.Sort Key1:=.[B2], Order1:=1, Key2:=.[C2], Order2:=1,Key3:=.[E2], Order3:=1,
Header:=xlYes
    '--Copia título.
    .Rows(1).Copy shRes.Rows(1)
    '--Configura intervalo inicial.
    Set rngStart = .[C2]
    Do Until IsEmpty(rngStart)
        Set rngPaste = shRes.Cells(shRes.Rows.Count, 1).End(xlUp).Offset(1)
        lngData = GetNextRange(rngStart)
        rngStart.Offset(, -2).Resize(, 5).Copy rngPaste

        'Copia para V1 toV14.
        rngStart.Offset(, 2).Resize(lngData).Copy
        rngPaste.Offset(, 5).PasteSpecial Paste:=xlAll, Operation:=xlNone,SkipBlanks:=False, Transpose:=True
        'Copia para V1FP para V14FP.
        rngStart.Offset(, 1).Resize(lngData).Copy
        rngPaste.Offset(, 19).PasteSpecial Paste:=xlAll, Operation:=xlNone,SkipBlanks:=False, Transpose:=True
        Set rngStart = rngStart.Offset(lngData)
    Loop
End With

Application.Goto shRes.[A1]
With shRes
    .Cells.Columns.AutoFit
    .Columns("D:E").Delete shift:=xlToLeft
End With

Application.ScreenUpdating = True
Application.CutCopyMode = False

If MsgBox("Você quer excluir a planilha original?", 36) = 6 Then
    Application.DisplayAlerts = False
    Sheets("TransposeData").Delete
    Application.DisplayAlerts = True
End If

Set rngPaste = Nothing
Set rngStart = Nothing
Set shRes = Nothing

Exit Sub

terminate:
End Sub

Function GetNextRange(ByVal rngSt As Range) As Long
    Dim i As Long
    i = 0

    Do Until rngSt.Value <> rngSt.Offset(i).Value
        i = i + 1
    Loop

    GetNextRange = i
End Function
```

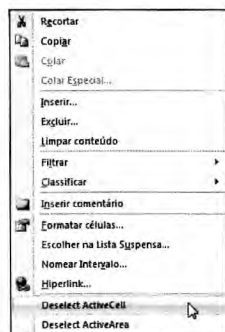
Selecionar/remover a seleção de células não-contíguas

Contribuição cedida por Tom Urtis.

Geralmente, para remover a seleção de uma única célula ou intervalo em uma planilha você deve clicar em uma célula não selecionada para remover a seleção de todas células e, então, recomeçar selecionando outra vez todas as células corretas. Isso é inconveniente caso você precise selecionar novamente muitas células não-contíguas.

Esse exemplo adiciona duas novas opções ao menu contextual de uma seleção: Remova a Seleção de ActiveCell e Remova a Seleção de ActiveArea. Com as células não-contíguas selecionadas, mantenha a tecla Ctrl pressionada, clique na célula da qual quer remover a seleção para ativá-la, solte a tecla Ctrl e depois clique com o botão direito do mouse na célula da qual quer remover a se-

leção. Será exibido o menu contextual mostrado na Figura 14.10. Clique no item do menu que remove a seleção daquela célula ativa ou da área contiguamente selecionada da qual ela faz parte.

Figura 14.10
O procedimento `ModifyRightClick` fornece um menu contextual personalizado para remover a seleção de células não contíguas.

Insira os seguintes procedimentos em um módulo padrão:

```
Sub ModifyRightClick()
'Adiciona as novas opções ao menu do clique com o botão direito do mouse.
Dim O1 As Object, O2 As Object

'Exclui as opções se já existirem.
On Error Resume Next
With CommandBars("Cell")
    .Controls("Deselect ActiveCell").Delete
    .Controls("Deselect ActiveArea").Delete
End With
On Error GoTo 0

'Adiciona as novas opções.
Set O1 = CommandBars("Cell").Controls.Add

With O1
    .Caption = "Deselect ActiveCell"
    .OnAction = "DeselectActiveCell"
End With

Set O2 = CommandBars("Cell").Controls.Add

With O2
    .Caption = "Deselect ActiveArea"
    .OnAction = "DeselectActiveArea"
End With

End Sub

Sub DeselectActiveCell()
Dim x As Range, y As Range

If Selection.Cells.Count > 1 Then
    For Each y In Selection.Cells
        If y.Address <> ActiveCell.Address Then
            If x Is Nothing Then
                Set x = y
            Else
                Set x = Application.Union(x, y)
            End If
        End If
    Next y
    If x.Cells.Count > 0 Then
        x.Select
    End If
End If

End Sub

Sub DeselectActiveArea()
Dim x As Range, y As Range
```

```
If Selection.Areas.Count > 1 Then
    For Each y In Selection.Areas
        If Application.Intersect(ActiveCell, y) Is Nothing Then
            If x Is Nothing Then
                Set x = y
            Else
                Set x = Application.Union(x, y)
            End If
        End If
    Next y
    x.Select
End If
End Sub
```

Adicione os seguintes procedimentos ao módulo ThisWorkbook:

```
Private Sub Workbook_Activate()
ModifyRightClick
End Sub

Private Sub Workbook_Deactivate()
Application.CommandBars("Cell").Reset
End Sub
```

Técnicas para profissionais do VBA

Os dez próximos utilitários me surpreendem. Nos vários fóruns de discussão na Internet, programadores do VBA aparecem constantemente com novas maneiras de fazer algo mais rápido ou melhor. Quando alguém posta algum novo código que, obviamente, supera o código que era considerado antes o melhor pela maioria, todos se beneficiam.

Drill-down de tabela dinâmica

Contribuição cedida por Tom Urtis.

O comportamento padrão de uma tabela dinâmica, quando você dá um clique duplo na seção de dados, é inserir uma nova planilha e exibir essas informações sobre drill-down na nova planilha.

O exemplo seguinte serve como uma opção conveniente para manter os recordsets explorados (*drilled-down*) na mesma planilha que a tabela dinâmica (veja Figura 14.11) e permitir que você os exclua como quiser. Para utilizar essa macro, dê um clique duplo na seção de dados ou na seção Totais para criar recordsets drill-down empilhados na próxima linha disponível dessa planilha. Para excluir todos os recordsets drill-down que você criou, dê um clique duplo em qualquer lugar em sua respectiva região atual.

Figura 14.11
Mostre o recordset drill-down na mesma planilha que a tabela dinâmica.

24	⊟Nancy	T3		3775	8424	8
25	Nancy Total			3775	8424	8
26	⊟Zelda	T4	86	1803	5037	
27	Zelda Total		86	1803	5037	
28	Total geral		48780	20396	38672	11738
29						
30	Nome	Região	Trimestre	Item	Cor	Vendas
31	Zelda	Leste	T4	Chapéus	Azul	86
32						

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
Application.ScreenUpdating = False
Dim LPTR&

With ActiveSheet.PivotTables(1).DataBodyRange
    LPTR = .Rows.Count + .Row - 1
End With

Dim PTT As Integer
On Error Resume Next
PTT = Target.PivotCell.PivotCellType
If Err.Number = 1004 Then
    Err.Clear
    If Not IsEmpty(Target) Then
        If Target.Row > Range("A1").CurrentRegion.Rows.Count + 1 Then
            Cancel = True
            With Target.CurrentRegion
                .Resize(.Rows.Count + 1).EntireRow.Delete
            End With
```

```
        End If
    Else
        Cancel = True
    End If
Else
    CS = ActiveSheet.Name
End If
Application.ScreenUpdating = True
End Sub
```

Configuração de página rápida

Contribuição cedida por Juan Pablo González Ruiz, de Bogotá, Colombia. Juan Pablo é o desenvolvedor da F&I Menu Wizard e trata todas as dúvidas de programação em espanhol do MrExcel.com.

Os próximos exemplos comparam os tempos de execução de variações na mudança de margens dos padrões para 1.5 polegadas e para o rodapé e cabeçalho para 1 polegada na Configuração de página. O programa de gravação de macros foi utilizado para criar Macro1. As macros 2, 3 e 4 mostram como o tempo de execução do código gravado pode ser diminuído. A Figura 14.12 mostra os resultados do teste de velocidade de cada variação.

Figura 14.12
Testes de velocidade da configuração de página.

	A	B	C	D
1	Macro1	Macro2	Macro3	Macro4
2	3,5702	0,8185	0,8149	0,1485
3	3,1218	0,8338	0,8257	0,1373
4	3,1133	0,8122	0,8165	0,1356
5	3,1374	0,8266	0,8296	0,1345
6	3,1579	0,8226	0,8228	0,1359
7	3,1158	0,8116	0,8205	0,1371
8	3,1363	0,8011	0,828	0,1373
9	3,1272	0,8111	0,8286	0,1343
10	3,1295	0,8093	0,8193	0,1334
11	3,1159	0,8106	0,8094	0,1334
12	3,1465	0,8026	0,8101	0,1335
13	3,1374	0,8269	0,8181	0,1385
14	3,1161	0,8274	0,8125	0,1367
15	3,1105	0,8249	0,8091	0,1373
16	3,1401	0,8227	0,8093	0,1381
17	3,1288	0,8065	0,825	0,1375
18	3,1908	0,8264	0,8264	0,1372
19	3,1788	0,8259	0,8223	0,1379
20	3,1602	0,8265	0,8248	0,1383
21	3,1184	0,8189	0,8324	0,1391
22	316%	82%	82%	14%
23	4	2	3	1

```
Sub Macro1()
'
'Macro Macro1
'Macro gravada em 28/3/2007
'
    Com ActiveSheet.PageSetup
        .PrintTitleRows = ""
        .PrintTitleColumns = ""
    End With
    ActiveSheet.PageSetup.PrintArea = ""
    With ActiveSheet.PageSetup
        .LeftHeader = ""
        .CenterHeader = ""
        .RightHeader = ""
        .LeftFooter = ""
        .CenterFooter = ""
        .RightFooter = ""
        .LeftMargin = Application.InchesToPoints(1)
        .RightMargin = Application.InchesToPoints(1)
        .TopMargin = Application.InchesToPoints(1)
        .BottomMargin = Application.InchesToPoints(1)
        .HeaderMargin = Application.InchesToPoints(1)
        .FooterMargin = Application.InchesToPoints(1)
        .PrintHeadings = False
        .PrintGridlines = False
        .PrintComments = xlPrintNoComments
        .PrintQuality = -3
        .CenterHorizontally = False
        .CenterVertically = False
        .Orientation = xlPortrait
        .Draft = False
        .PaperSize = xlPaperLetter
        .FirstPageNumber = 1
        .Order = xlDownThenOver
        .BlackAndWhite = False
```

```
            .Zoom = False
            .FitToPagesWide = 1
            .FitToPagesTall = 1
            .PrintErrors = xlPrintErrorsDisplayed
            .OddAndEvenPagesHeaderFooter = False
            .DifferentFirstPageHeaderFooter = False
            .ScaleWithDocHeaderFooter = True
            .AlignMarginsHeaderFooter = False
            .EvenPage.LeftHeader.Text = ""
            .EvenPage.CenterHeader.Text = ""
            .EvenPage.RightHeader.Text = ""
            .EvenPage.LeftFooter.Text = ""
            .EvenPage.CenterFooter.Text = ""
            .EvenPage.RightFooter.Text = ""
            .FirstPage.LeftHeader.Text = ""
            .FirstPage.CenterHeader.Text = ""
            .FirstPage.RightHeader.Text = ""
            .FirstPage.LeftFooter.Text = ""
            .FirstPage.CenterFooter.Text = ""
            .FirstPage.RightFooter.Text = ""
        End With
    End Sub
```

O programa de gravação de macros está fazendo muito trabalho extra, o que requer mais tempo de processamento. Considerando isso, mais o fato que o objeto `PageSetup` é um dos mais lentos de se atualizar, você pode realmente ter um grande problema. Então, eis uma versão mais limpa, que utiliza apenas a tecla Delete:

```
Sub Macro1_Version2()
    With ActiveSheet.PageSetup
        .LeftMargin = Application.InchesToPoints(1.5)
        .RightMargin = Application.InchesToPoints(1.5)
        .TopMargin = Application.InchesToPoints(1.5)
        .BottomMargin = Application.InchesToPoints(1.5)
        .HeaderMargin = Application.InchesToPoints(1)
        .FooterMargin = Application.InchesToPoints(1)
    End With
End Sub
```

Certo, essa é mais rápida que a Macro1 (a redução média está em torno de 70 por cento em alguns testes simples!), mas pode ser ainda mais aprimorada. Como observado anteriormente, o objeto `PageSetup` leva um longo tempo de processamento. Se você reduzir o número de operações que o VBA tem de fazer e incluir algumas funções `IF` para atualizar somente as propriedades que requerem alteração, você poderá obter resultados muitos melhores.

No caso a seguir, a função `Application.InchesToPoints` foi codificada manualmete com o valor `inches`. A terceira versão de Macro1 é semelhante a:

```
Sub Macro1_Version3()
    With ActiveSheet.PageSetup
        If .LeftMargin <> 108 Then .LeftMargin = 108
        If .RightMargin <> 108 Then .RightMargin = 108
        If .TopMargin <> 108 Then .TopMargin = 108
        If .BottomMargin <> 108 Then .BottomMargin = 108
        If .HeaderMargin <> 72 Then .HeaderMargin = 72
        If .FooterMargin <> 72 Then .FooterMargin = 72
    End With
End Sub
```

Você deve ver a diferença disso quando não estiver alterando todas as margens-padrão.

Outra opção pode reduzir o tempo de execução em mais de 95 por cento! Ela utiliza o método XLM `PAGE.SETUP`. Os parâmetros necessários são `left`, `right`, `top`, `bot`, `head_margin` e `foot_margin`. Eles são medidos em polegadas, não pontos. Então, utilizando as mesmas margens que já vínhamos usando, uma quarta versão de Macro1 é semelhante a esta:

```
Sub Macro1_Version4()
    Dim St As String
    St = "PAGE.SETUP(, , " &"1.5, 1.5, 1.5, 1.5" &", 0, False, False, False, 1, 1, True, 1, 1,False, , " &"1, 1"
&", False)"
    Application.ExecuteExcel4Macro St
End Sub
```

A segunda e a quarta linhas de St correspondem a esses parâmetros. Mas você precisa seguir algumas precauções simples. Primeiro, essa macro conta com a linguagem XLM, que ainda é incluída no Excel para permitir a compatibilidade com as versões anteriores, mas não sabemos quando a Microsoft irá lançá-la. Segundo, tenha cuidado ao configurar os parâmetros de PAGE.SETUP, porque se um deles estiver errado PAGE.SETUP não será executado e não gerará erro, o que possivelmente deixará a configuração de página errada.

Calculando o tempo para executar o código

Você pode se perguntar como calcular tempo passado a um milésimo de segundo, como mostrado anteriormente na Figura 14.12.

Esse é o código utilizado nesta seção para gerar os resultados de tempo para as macros:

```
Public Declare Function QueryPerformanceFrequencyLib "kernel32" (lpFrequency As Currency) As Long
Public Declare Function QueryPerformanceCounterLib "kernel32.dll" (lpPerformanceCount As Currency) As Long

Sub CalculateTime()
    Dim Ar(1 To 20, 1 To 4) As Currency, WS As Worksheet
    Dim n As Currency, str As Currency, fin As Currency
    Dim y As Currency

    Dim i As Long, j As Long

    Application.ScreenUpdating = False
    For i = 1 To 4
        For j = 1 To 20
            Set WS = ThisWorkbook.Sheets.Add
            WS.Range("A1").Value = 1
            QueryPerformanceFrequency y
            QueryPerformanceCounter str
            Select Case i
            Case 1: Macro1
            Case 2: Macro1_Version2
            Case 3: Macro1_Version3
            Case 4: Macro1_Version4
            End Select
            QueryPerformanceCounter fin
            Application.DisplayAlerts = False
            WS.Delete
            Application.DisplayAlerts = True
            n = (fin - str)
            Ar(j, i) = CCur(Format(n, "##########.############") / y)
        Next j
    Next i
    With Range("A1").Resize(1, 4)
        .Value = Array("Macro1", "Macro2", "Macro3", " Macro4")
        .Font.Bold = True
    End With
    Range("A2").Resize(20, 4).Value = Ar

    With Range("A22").Resize(1, 4)
        .FormulaR1C1 = "=AVERAGE(R2C:R21C)"
        .Offset(1).FormulaR1C1 = "=RANK(R22C,R22C1:R22C4,1)"
        .Resize(2).Font.Bold = True
    End With
    Application.ScreenUpdating = True
End Sub
```

Personalizando a ordem de classificação

Contribuição cedida por Wei Jiang, da cidade de Shiyan, China. Jiang é consultor do MrExcel.com.

Por padrão, o Excel permite classificar listas numérica ou alfabeticamente, mas às vezes não é disso que precisamos. Por exemplo, um cliente poderia precisar dos dados de vendas classificados diariamente pela ordem de divisão padrão de cintos, bolsas, relógios, carteiras e qualquer outro produto. Esse exemplo utiliza uma lista personalizada de ordem de classificação para classificar um intervalo de dados em ordem de divisão padrão. A Figura 14.13 mostra os resultados.

Figura 14.13
Quando você utiliza a macro, a lista em A:C é classificada primeiro por data, depois pela lista de classificação personalizada na Coluna I.

	A	B	C	D	E	F	G	H	I	J
1	Data	Categoria	nº vendido						Cintos	
2	01/01/2007	Cintos	15						Carteiras	
3	01/01/2007	Carteiras	23						Relógios	
4	01/01/2007	Relógios	42						Bolsas	
5	01/01/2007	Bolsas	17						Todos os demais	
6	01/01/2007	Todos os demais	36							
7	02/01/2007	Cintos	17							
8	02/01/2007	Carteiras	21							
9	02/01/2007	Relógios	43							
10	02/01/2007	Bolsas	18							
11	02/01/2007	Todos os demais	42							
12	03/01/2007	Cintos	21							
13	03/01/2007	Carteiras	20							
14	03/01/2007	Relógios	35							
15	03/01/2007	Bolsas	19							
16	03/01/2007	Todos os demais	45							
17										

```
Sub CustomSort()

    ' Adiciona a lista personalizada a Listas Personalizadas.
    Application.AddCustomList ListArray:=Range("I1:I5")

    ' Obtém o número de lista.
    nIndex = Application.GetCustomListNum(Range("I1:I5").Value)

    ' Agora, poderíamos classificar um intervalo com a lista personalizada.
    ' Nota, devemos utilizar nIndex+1 como o número personalizado de lista aqui,
    ' para o primeiro a ordem é Normal.
    Range("A2:C16").Sort Key1:=Range("B2"), Order1:=xlAscending,Header:=xlNo, Orientation:=xlSortColumns,OrderCu
stom:=nIndex + 1
    Range("A2:C16").Sort Key1:=Range("A2"), Order1:=xlAscending,Header:=xlNo, Orientation:=xlSortColumns

    ' No fim, devemos remover essa lista personalizada...
    Application.DeleteCustomList nIndex
End Sub
```

Indicador de progresso de célula

Contribuição cedida por Tom Urtis.

Tenho de admitir que as novas opções de formatação condicional no Excel, como barras de dados, são fantásticas. Mas ainda não há opção para um visual como aquele mostrado na Figura 14.14. O exemplo seguinte cria um indicador de progresso na Coluna C baseado em entradas nas Colunas A e B.

Figura 14.14
Utilize indicadores em células para mostrar progresso.

	A	B	C
1	Progresso feito	Progresso exigido	Representação visual de Progresso feito vs Progresso exigido
2	11	15	■■■■■■■■■■■□□□□
3	14	20	■■■■■■■■■■■■■■□□□□□□
4	1	5	■□□□□
5	4	10	■■■■□□□□□□
6	4	10	■■■■□□□□□□
7	10	10	■■■■■■■■■■
8	8	10	■■■■■■■■□□
9		10	□□□□□□□□□□
10		10	□□□□□□□□□□
11		10	□□□□□□□□□□
12			

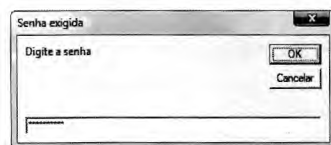
```
Private Sub Worksheet_Change(ByVal Target As Range)
If Target.Column > 2 Or Target.Cells.Count > 1 Then Exit Sub
If Application.IsNumber(Target.Value) = False Then
    Application.EnableEvents = False
    Application.Undo
    Application.EnableEvents = True
    MsgBox "Somente números por favor."
    Exit Sub
End If
Select Case Target.Column
    Case 1
        If Target.Value > Target.Offset(0, 1).Value Then
            Application.EnableEvents = False
            Application.Undo
            Application.EnableEvents = True
            MsgBox "O valor na coluna A pode não ser maior do que o valor na coluna B."
            Exit Sub
        End If
    Case 2
        If Target.Value < Target.Offset(0, -1).Value Then
            Application.EnableEvents = False
            Application.Undo
```

```
            Application.EnableEvents = True
            MsgBox "O valor na coluna B pode não ser menor  " &"do que o valor na coluna A."
            Exit Sub
        End If
End Select
Dim x As Long
x = Target.Row
Dim z As String
z = Range("B" & x).Value - Range("A" & x).Value
With Range("C" & x)
    .Formula = "=IF(RC[-1]<=RC[-2],REPT(""n"",RC[-1])&REPT(""n"",RC[-2]-RC[-1]),REPT(""n"",RC[-
2])&REPT(""o"",RC[-1]-RC[-2]))"
    .Value = .Value
    .Font.Name = "Wingdings"
    .Font.ColorIndex = 1
    .Font.Size = 10
    If Len(Range("A" & x)) <> 0 Then
        .Characters(1, (.Characters.Count - z)).Font.ColorIndex = 3
        .Characters(1, (.Characters.Count - z)).Font.Size = 12
    End If
End With
End Sub
```

Caixa protegida por senha

Contribuição cedida por Daniel Klann, de Sydney, Austrália. Daniel trabalha principalmente com o VBA no Excel e Access, mas, pode eventualmente trabalhar em todos os tipos de linguagem. Ele mantém um site Web em www. danielklann.com.

Utilizar uma caixa de entrada para proteção por senha tem um grande defeito de segurança: os caracteres inseridos são facilmente visualizáveis. O programa a seguir transforma os caracteres em asteriscos à medida que eles são inseridos — exatamente como um campo de senha real (veja Figura 14.15).

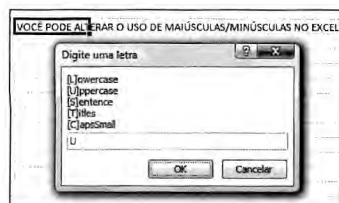
Figura 14.15
Utilize uma caixa de entrada como um campo de senha seguro.

```
Private Declare Function CallNextHookEx Lib "user32" (ByVal hHook As Long, _
ByVal ncode As Long, ByVal wParam As Long, lParam As Any) As Long

Private Declare Function GetModuleHandle Lib "kernel32"Alias "GetModuleHandleA" (ByVal lpModuleName As String) As
Long

Private Declare Function SetWindowsHookEx Lib "user32"Alias "SetWindowsHookExA"(ByVal idHook As Long, ByVal lpfn
As Long,ByVal hmod As Long,ByVal dwThreadId As Long) As Long

Private Declare Function UnhookWindowsHookEx Lib "user32"(ByVal hHook As Long) As Long

Private Declare Function SendDlgItemMessage Lib "user32"Alias "SendDlgItemMessageA"(ByVal hDlg As Long,ByVal
nIDDlgItem As Long, ByVal wMsg As Long,ByVal wParam As Long, ByVal lParam As Long) As Long

Private Declare Function GetClassName Lib "user32"Alias "GetClassNameA" (ByVal hwnd As Long,ByVal lpClassName As
String,ByVal nMaxCount As Long) As Long

Private Declare Function GetCurrentThreadIdLib "kernel32" () As Long

'Constantes a ser usadas em nossas funções de API
Private Const EM_SETPASSWORDCHAR = &HCC
Private Const WH_CBT = 5
Private Const HCBT_ACTIVATE = 5
Private Const HC_ACTION = 0

Private hHook As Long

Public Function NewProc(ByVal lngCode As Long,ByVal wParam As Long, ByVal lParam As Long) As Long
    Dim RetVal
    Dim strClassName As String, lngBuffer As Long
```

```
        If lngCode < HC_ACTION Then
            NewProc = CallNextHookEx(hHook, lngCode, wParam, lParam)
            Exit Function
        End If

        strClassName = String$(256, " ")
        lngBuffer = 255

        If lngCode = HCBT_ACTIVATE Then    'Uma janela foi ativada.

            RetVal = GetClassName(wParam, strClassName, lngBuffer)

            'Verifica o nome de classe da Inputbox.
            If Left$(strClassName, RetVal) = "#32770" Then
                'Altera o controle de edição para exibir o caractere de senha *.
                'Você pode alterar o Asc("*") como quiser.
                SendDlgItemMessage wParam, &H1324, EM_SETPASSWORDCHAR, Asc("*"), &H0
            End If

        End If

        'Essa linha assegurará que quaisquer outros hooks que possam ser definidos sejam
        'chamados corretamente.
        CallNextHookEx hHook, lngCode, wParam, lParam

    End Function

    Public Function InputBoxDK(Prompt, Optional Title,Optional Default, Optional XPos, _
    Optional YPos, Optional HelpFile, Optional Context) As String
        Dim lngModHwnd As Long, lngThreadID As Long

        lngThreadID = GetCurrentThreadId
        lngModHwnd = GetModuleHandle(vbNullString)

        hHook = SetWindowsHookEx(WH_CBT, AddressOf NewProc, lngModHwnd, lngThreadID)
        On Error Resume Next
        InputBoxDK = InputBox(Prompt, Title, Default, XPos, YPos, HelpFile, Context)
        UnhookWindowsHookEx hHook

    End Function

    Sub PasswordBox()
    If InputBoxDK("Digite a senha", "Senha exigida") <> "senha" Then
            MsgBox "Essa não é uma senha correta."
        Else
            MsgBox "Senha correta!  Entre."
        End If
    End Sub
```

Maiúsculas e minúsculas

Contribuição cedida Por Ivan F. Moala.

Uma palavra pode mudar de 'caixa' — maneira de dizer que a palavra é composta de letras maiúsculas [caixa alta] ou minúsculas [caixa baixa] —, mas, surpreendentemente, o Excel não tem essa capacidade. O programa a seguir permite ao usuário do Excel alterar a caixa do texto em qualquer intervalo selecionado, como mostrado na Figura 14.16.

Figura 14.16 Agora é possível alterar palavras maiúsculas ou minúsculas, exatamente como no Word.

```
Sub TextCaseChange()
Dim RgText As Range
Dim oCell As Range
Dim Ans As String
Dim strTest As String
```

```
Dim sCap As Integer,lCap As Integer,i As Integer

'// Primeiro você precisa selecionar um intervalo a ser alterado!

Again:
Ans = Application.InputBox("[L]owercase" & vbCr & "[U]ppercase" & vbCr &"[S]entence" & _
    vbCr & "[T]itles" & vbCr & "[C]apsSmall","Type in a Letter", Type:=2)

If Ans = "False" Then Exit Sub
If InStr(1, "LUSTC", UCase(Ans), vbTextCompare) = 0Or Len(Ans) > 1 Then GoTo Again

On Error GoTo NoText
If Selection.Count = 1 Then
    Set RgText = Selection
Else
    Set RgText = Selection.SpecialCells(xlCellTypeConstants, 2)
End If
On Error GoTo 0

For Each oCell In RgText
    Select Case UCase(Ans)
        Case "L": oCell = LCase(oCell.Text)
        Case "U": oCell = UCase(oCell.Text)
        Case "S": oCell = UCase(Left(oCell.Text, 1)) &LCase(Right(oCell.Text, Len(oCell.Text) - 1))
        Case "T": oCell = Application.WorksheetFunction.Proper(oCell.Text)
        Case "C"
                lCap = oCell.Characters(1, 1).Font.Size
                sCap = Int(lCap * 0.85)
                'Caixa-baixa para tudo.
                oCell.Font.Size = sCap
                oCell.Value = UCase(oCell.Text)
                strTest = oCell.Value
                'Caixa-alta para a primeira das palavras.
                strTest = Application.Proper(strTest)
                For i = 1 To Len(strTest)
                    If Mid(strTest, i, 1) = UCase(Mid(strTest, i, 1)) Then
                        oCell.Characters(i, 1).Font.Size = lCap
                    End If
                Next i
    End Select
Next

Exit Sub
NoText:
MsgBox "Nenhum texto em sua seleção @ " & Selection.Address

End Sub
```

Selecionando com `SpecialCells`

Contribuição cedida por Ivan F. Moala.

Em geral, quando você quer encontrar certos valores, textos ou fórmulas em um intervalo, o intervalo é selecionado e todas as células são testadas. O próximo exemplo mostra como `SpecialCells` pode ser utilizado para selecionar somente as células desejadas. Ter menos células para verificar acelera o código:

```
Sub SpecialRange()
Dim TheRange As Range
Dim oCell As Range

    Set TheRange = Range("A1:Z200").SpecialCells(__
    xlCellTypeConstants, xlTextValues)

    For Each oCell In TheRange
        If oCell.Text = "Seu texto" Then
            MsgBox oCell.Address
            MsgBox TheRange.Cells.Count
        End If
    Next oCell

End Sub
```

Menu ActiveX para evento de clicar com o botão direito do mouse

Não há menu predefinido para o evento de clicar com o botão direito do mouse de objetos ActiveX em uma planilha. O programa a seguir é para isso, utilizando um botão de comando para o exemplo na Figura 14.17. Configure a propriedade `Take Focus on Click` do botão de comando como `False`.

Figura 14.17
Personalize o menu contextual (clique com o botão direito do mouse) de um controle ActiveX.

Clique-me com o botão direito para uma demonstração do menu personalizado.

Minha Macro 1
Minha Macro 2
Minha Macro 3

Coloque o seguinte no módulo ThisWorkbook:

```
Private Sub Workbook_Open()
With Application
    .CommandBars("Cell").Reset
    .WindowState = xlMaximized
    .Goto Sheet1.Range("A1"), True
End With
End Sub

Private Sub Workbook_Activate()
Application.CommandBars("Cell").Reset
End Sub

Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object,ByVal Target As Range, Cancel As Boolean)
Application.CommandBars("Cell").Reset
End Sub

Private Sub Workbook_Deactivate()
Application.CommandBars("Cell").Reset
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
With Application
    .CommandBars("Cell").Reset
    .WindowState = xlMaximized
    .Goto Sheet1.Range("A1"), True
End With
ThisWorkbook.Save
End Sub
```

Coloque o seguinte em um módulo padrão:

```
Sub MyRightClickMenu()
Application.CommandBars("Cell").Reset
Dim cbc As CommandBarControl
  For Each cbc In Application.CommandBars("cell").Controls
      cbc.Visible = False
  Next cbc
With Application.CommandBars("Cell").Controls.Add(temporary:=True)
    .Caption = "Minha Macro 1"
    .OnAction = "Test1"
   End With
With Application.CommandBars("Cell").Controls.Add(temporary:=True)
    .Caption = "Minha Macro 2"
    .OnAction = "Test2"
   End With
With Application.CommandBars("Cell").Controls.Add(temporary:=True)
    .Caption = "Minha Macro 3"
    .OnAction = "Test3"
   End With
Application.CommandBars("Cell").ShowPopup
End Sub

Sub Test1()
MsgBox "Essa é a macro Teste1 do menu de evento do clique " & "com o botão direito personalizado do _
    objeto ActiveX..", , "Ítem de menu ''Minha Macro 1''"
End Sub

Sub Test2()
```

```
MsgBox "Essa é a macro Teste2 do menu de evento do clique" & "com o botão direito personalizado do _
    objeto ActiveX.", , "Ítem de menu ''Minha Macro 2''"
End Sub

Sub Test3()
MsgBox "Essa é a macro Teste3 do menu de evento do clique" & "com o botão direito personalizado do_
    objeto ActiveX.", , "Ítem de menu ''Minha Macro 3''"
End Sub
```

Aplicativos interessantes

Os exemplos a seguir são aplicativos interessantes que você pode incorporar em seus próprios projetos.

Cotações de ações/financeiras históricas

Contribuição cedida por Nathan P. Oliver.

O programa seguinte recupera a média de um registrador de cotações válido ou o fechamento de grandes ações para a data especificada (veja Figura 14.18).

	A	B	C
1	Símbolo	Data	Média/Fechar
2	Dell	12/01/1994	25,3125
3	MSFT	30/01/2003	49,18
4	VFINX	20/01/2000	
5	INNDX	06/01/2003	
6	INSTX	17/02/2004	

Figura 14.18
Recupera informações sobre ações.

```
Private Sub GetQuote()
Dim ie As Object, lCharPos As Long, sHTML As String
Dim HistDate As Date, HighVal As String, LowVal As String
Dim cl As Range

Set cl = ActiveCell
HistDate = cl(, 0)

If Intersect(cl, Range("C2:C" & Cells.Rows.Count)) Is Nothing Then
    MsgBox "Você precisa selecionar uma célula na coluna C."
    Exit Sub
End If

If Not CBool(Len(cl(, -1))) Or Not CBool(Len(cl(, 0))) Then
    MsgBox "Você precisa inserir um símbolo ou data."
    Exit Sub
End If

Set ie = CreateObject("InternetExplorer.Application")

With ie
    .Navigatehttp://bigcharts.marketwatch.com/historical &"/default.asp?detect=1&symbol="& cl(, -1) & _
        "&close_date=" & Month(HistDate) & "%2F" &Day(HistDate) & "%2F" & Year(HistDate) & "&x=31&y=26"
    Do While .Busy And .ReadyState <> 4
        DoEvents
    Loop
    sHTML = .Document.body.innertext
    .Quit
End With

Set ie = Nothing

lCharPos = InStr(1, sHTML, "High:", vbTextCompare)
If lCharPos Then HighVal = Mid$(sHTML, lCharPos + 5, 15)

If Not Left$(HighVal, 3) = "n/a" Then
    lCharPos = InStr(1, sHTML, "Low:", vbTextCompare)
    If lCharPos Then LowVal = Mid$(sHTML, lCharPos + 4, 15)
    cl.Value = (Val(LowVal) + Val(HighVal)) / 2
Else: lCharPos = InStr(1, sHTML, "Closing Price:", vbTextCompare)
    cl.Value = Val(Mid$(sHTML, lCharPos + 14, 15))
End If

Set cl = Nothing
End Sub
```

Utilizando a extensibilidade do VBA para adicionar código a novas pastas de trabalho

Você tem uma macro que move dados para uma nova pasta de trabalho para os gerentes regionais. E se você também tiver de copiar macros para a nova pasta de trabalho? Você pode utilizar a Visual Basic for Application Extensibility para importar módulos para uma pasta de trabalho ou realmente escrever linhas de código para a pasta de trabalho.

Para utilizar qualquer um desses exemplos, você deve primeiro abrir o Editor do VB, escolher Referências no menu Ferramentas e selecionar a referência para Microsoft Visual Basic for Applications Extensibility 5.3. Você também deve confiar o acesso ao VBA na guia Desenvolvedor, escolhendo Segurança de Macro, e marcando Confiar Acesso ao Modelo de Objeto do Projeto do VBA.

A maneira mais fácil de utilizar a Extensibilidade do VBA é exportar um módulo ou userform completo a partir do projeto atual e importá-lo para a nova pasta de trabalho. Talvez você tenha um aplicativo com milhares de linhas de código. Você quer criar uma nova pasta de trabalho com dados para o gerente regional e fornecer-lhe suas três macros para permitir formatação e impressão personalizadas. Coloque todas essas macros em um módulo chamado ModToRegion. As macros nesse módulo também chamam o userform frmRegion. O código seguinte transfere esse código da pasta de trabalho atual para a nova pasta de trabalho:

```
Sub MoveDataAndMacro()
    Dim WSD as worksheet
    Set WSD = Worksheets("Report")
    ' Copia o relatório para uma nova pasta de trabalho.
    WSD.Copy
    ' A pasta de trabalho ativa é agora a nova pasta de trabalho.
    ' Exclui qualquer cópia antiga do módulo a partir de C.
    On Error Resume Next
    ' Exclui todas as cópias extraviadas da unidade de disco.
    Kill ("C:\ModToRegion.bas")
    Kill ("C:\frmRegion.frm")
    On Error GoTo 0
    ' Exporta módulo & formulário dessa pasta de trabalho.
    ThisWorkbook.VBProject.VBComponents("ModToRegion").Export("C:\ModToRegion.bas")
    ThisWorkbook.VBProject.VBComponents("frmRegion").Export ("C:\frmRegion.frm")
    ' Importa para a nova pasta de trabalho.
    ActiveWorkbook.VBProject.VBComponents.Import ("C:\ModToRegion.bas")
    ActiveWorkbook.VBProject.VBComponents.Import ("C:\frmRegion.frm")
    On Error Resume Next
    Kill ("C:\ModToRegion.bas")
    Kill ("C:\frmRegion.bas")
    On Error GoTo 0
End Sub
```

O método anterior funcionará se você precisar mover módulos ou userforms para uma nova pasta de trabalho. Mas e se você precisar escrever algum código para a macro Workbook_Open no módulo ThisWorkbook? Duas ferramentas podem ser utilizadas. O método `Lines` permitirá retornar um conjunto particular de linhas de código de um dado módulo. O método `InsertLines` permitirá inserir linhas de código para um novo módulo.

ATENÇÃO

A cada chamada para `InsertLines`, você deve inserir uma macro completa. O Excel tentará compilar o código depois de cada chamada para `InsertLines`. Se você inserir linhas que não compilam completamente, o Excel poderá travar com uma GPF (General Protection Fault).

```
Sub MoveDataAndMacro()
    Dim WSD as worksheet
    Dim WBN as Workbook
    Dim WBCodeMod1 As Object, WBCodeMod2 As Object
    Set WSD = Worksheets("Report")
    ' Copie o relatório para uma nova pasta de trabalho.
    WSD.Copy
    ' A pasta de trabalho ativa é agora a nova pasta de trabalho.
    Set WBN = ActiveWorkbook
    ' Copie as rotinas de tratamento de evento no nível da pasta de trabalho.
    Set WBCodeMod1 = ThisWorkbook.VBProject.VBComponents("ThisWorkbook").CodeModule
    Set WBCodeMod2 = WBN.VBProject.VBComponents("ThisWorkbook").CodeModule
    WBCodeMod2.insertlines 1, WBCodeMod1.Lines(1, WBCodeMod1.countoflines)
End Sub
```

Próximos passos

O Excel 2007 oferece novas ferramentas de visualização de dados fantásticas, como barras de dados, escalas de cor, conjuntos de ícone e regras de formatação condicional aprimoradas. No Capítulo 15, "Visualizações de dados e formatação condicional", você aprenderá a automatizar as novas ferramentas e usar o VBA para invocar escolhas não-disponíveis na interface com o usuário Excel.

NOVO Visualizações de dados e formatação condicional

15

Introdução às visualizações de dados

As ferramentas de visualização de dados no Excel 2007 representam um de seus melhores recursos novos. A Microsoft adicionou uma nova camada de desenho que pode armazenar conjuntos de ícones, barras de dados e escalas de cor. Diferentemente dos gráficos SmartArt, a Microsoft expôs o modelo inteiro de objeto para as ferramentas de visualização de dados, então você pode utilizar o VBA para adicionar visualizações de dados aos seus relatórios.

O Excel 2007 fornece uma variedade de novas visualizações de dados. Uma descrição de cada visualização aparece a seguir, com um exemplo mostrado na Figura 15.1.

- **Barras de dados** — A barra de dados adiciona um gráfico de barras na célula a cada célula em um intervalo. Os números maiores têm as barras maiores e os menores, as barras menores. Você pode controlar a cor das barras bem como os valores que elas devem receber.
- **Escalas de cor** — O Excel aplica uma cor a cada célula, de gradiente de duas ou três cores. Os gradientes de duas cores são melhores para relatórios que serão apresentados no modo monocromático. Já os de três cores exigem apresentação colorida, mas podem representar um relatório com uma combinação das cores vermelho/verde/amarelo de um semáforo tradicional. Você pode controlar os pontos ao longo do *continuum* em que cada cor inicia e controlar as duas ou as três cores.
- **Conjuntos de ícones** — O Excel atribui um ícone a cada número. Os conjuntos de ícones podem conter três ícones (como a luzes vermelhas, amarelas e verdes do semáforo), quatro ou cinco ícones (como as barras de energia do telefone celular). Com os conjuntos de ícones, você pode controlar os limites numéricos de cada ícone, inverter a ordem deles ou escolher apenas exibi-los.
- **Acima/abaixo da média** — Localizada no menu flutuante de regras a superior/inferior, essas regras facilitam realçar todas as células que estão acima da média. É possível escolher a formatação que deve ser aplicada às células. Note na coluna G da Figura 15.1 que apenas 30 por cento das células estão acima da média. Contraste com os Primeiros 50 Por Cento na coluna I.
- **Regras de Primeiros/Últimos** — O Excel realça *n* por cento das primeiras e últimas de células ou realça as primeiras e últimas *n* células em um intervalo.
- **Valores duplicados** — O Excel realça quaisquer valores que sejam repetidos em um conjunto de dados. Como o novo comando Remover Duplicatas na guia Dados da faixa é muito destrutivo, você pode preferir realçar as duplicatas e, inteligentemente, decidir que registros quer remover.
- **Realçar regras de células** — As regras de formatação condicional de legado, como Maior Que, Menor Que, Está Entre e Texto Que Contém, estão ainda

NESTE CAPÍTULO

disponíveis no Excel 2007. As poderosas condições Fórmula também estão disponíveis, embora você venha a utilizá-las com menos freqüência com a adição das regras de primeiros/últimos itens da média.

Figura 15.1 Visualizações como barras de dados, escalas de cor, conjunto de ícones e regras de primeiros/últimos itens são controladas na interface com o usuário do Excel na lista suspensa Formatação Condicional na guia Início da faixa.

Novos métodos e propriedades do VBA para visualizações de dados

Todas as configurações de visualização de dados são gerenciadas no VBA com a coleção FormatConditions. O Excel 2007 acrescenta sete novos métodos de aplicar condições, como o `AddDataBar`, `AddIconSet`, `AddTop10`, `AddUniqueValues` e assim por diante.

No Excel 2007, é possível adicionar várias condições de formatação condicional diferentes ao mesmo intervalo. Por exemplo, você pode aplicar uma escala de cor de duas cores, um conjunto de ícones e uma barra de dados ao mesmo intervalo. O Excel 2007 adiciona uma propriedade `Priority` para especificar as condições que devem ser calculadas primeiro. Métodos como `SetFirstPriority` e `SetLastPriority` asseguram que uma nova condição de formato seja executada antes ou depois de outras.

A propriedade `StopIfTrue` funciona em conjunto com a propriedade `Priority`. Na seção "Utilizando truques de visualização" mais adiante neste capítulo, você verá como utilizar a propriedade `StopIfTrue` em uma condição fictícia para que outra formatação seja aplicada apenas a certos subconjuntos de um intervalo.

A propriedade `Type` foi significativamente expandida no Excel 2007. Embora essa propriedade fosse inicialmente um alternador entre `CellValue` e `Expression`, 13 novos tipos foram adicionados no Excel 2007. A Tabela 15.1 mostra os valores válidos para a propriedade `Type`. Os itens 3 a 18 são novos no Excel 2007.

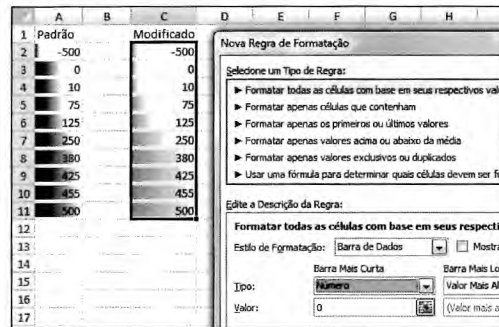
Tabela 15.1 Tipos válidos para uma condição de formato

Valor	Descrição	Constantes dos VBA
1	Valor da célula	`xlCellValue`
2	Expressão	`xlExpression`
3	Escala de cor	`xlColorScale`
4	Barras de dados	`xlDatabar`
5	10 Primeiros valores	`xlTop10`
6	Conjunto de ícones	`XlIconSet`
8	Valores únicos	`xlUniqueValues`
9	String de texto	`xlTextString`
10	Condição de lacunas	`xlBlanksCondition`
11	Período	`xlTimePeriod`
12	Condição acima da média	`xlAboveAverageCondition`
13	Condição sem lacunas	`xlNoBlanksCondition`
16	Condição de erros	`xlErrorsCondition`
17	Condição sem erros	`xlNoErrorsCondition`
18	Compare colunas	`xlCompareColumns`

Adicionando as barras de dados a um intervalo

O comando Barra de Dados adiciona um gráfico de barras de dados na célula para cada célula em um intervalo. Em geral, os valores menores no conjunto de dados recebem uma barra de quatro pixels de largura. Os valores mais altos no conjunto de dados recebem uma barra que tem 90 por cento da largura da célula.

Na Figura 15.2, um valor de `-500` na célula A2 faz com que valores baixos, como 0 e 10, nas células A3:A4 tenham uma barra dados relativamente mais longa. Utilizando a caixa de diálogo Editar Regra de Formatação, você pode especificar que qualquer valor 0 ou menor deve ter a menor barra de dados. Na coluna C, o tamanho de cada barra de dados representa melhor os valores esperados de 0 a 500.

Figura 15.2
Se seu conjunto de dados tiver outliers, você pode ajustar as regras da barra de dados para especificar um certo valor que deve obter a maior ou a menor barra de dados. Aqui, a barra mais curta foi mudada do menor valor para zero.

Utilize o método `FormatConditions.AddDataBar` para adicionar um novo membro `FormatCondition` à coleção `FormatConditions` de um intervalo:

```
Range("A1:A10").FormatConditions.AddDatabar
```

Como não é possível afirmar que você se tem apenas uma condição aplicada a um intervalo, refira-se à nova condição de barra de dados utilizando a propriedade `Count`:

```
ThisCond = Range("A1:A10").FormatConditions.Count
```

Se quiser certificar-se de que esse é a única condição de formato aplicada ao intervalo, use o método `FormatConditions.Delete`:

```
Rang Range("A1:A10").FormatConditions.Delete
```

Especifique uma cor para a barra de dados utilizando as propriedades `Color` and `TintAndShade` para BarColor. A propriedade `Color` pode ser qualquer uma das 16 milhões de cores. Defina-a utilizando a função `RGB`. A propriedade `TintAndShade` modificará a cor selecionada. Especifique um valor de -1 (mais escuro) para 1 (mais claro). O código a seguir muda a cor da barra de dados para vermelho e a torna mais escura que o normal:

```
With Range("A1:A10").FormatConditions(ThisCond).BarColor
    .Color = RGB(255, 0, 0) ' Vermelho
    .TintAndShade = -0.5 ' Mais escuro que o normal
End With
```

Por padrão, o Excel atribui a menor barra de dados ao valor mínimo e a barra de dados mais longa ao valor máximo. Se você quiser sobrescrever os padrões, utilize o método Modify para as propriedades MinPoint ou MaxPoint. Especifique um tipo a partir dos mostrados na Tabela 15.2. Os tipos 0, 3, 4 e 5 requerem um valor. A Tabela 15.2 mostra os tipos válidos.

Tabela 15.2 Tipos MinPoint e MaxPoint

Valor	Descrição	Constantes dos VBA
0	O número é utilizado.	`xlConditionNumber`
1	Valor mais baixo da lista de valores.	`xlConditionValueLowest Value`
2	Valor mais alto da lista de valores.	`xlConditionValueHighestValue`
3	A percentagem é utilizada.	`xlConditionValuePercent`
4	A fórmula é utilizada.	`xlConditionValueFormula`
5	O percentil é utilizado.	`xlConditionValuePercentile`
-1	Sem valor condicional.	`xlConditionValueNone`

Para que a menor barra seja atribuída a valores de 0 e abaixo de 0, utilize este código:

```
Range("A1:A10").FormatConditions(ThisCond).MinPoint.ModifyNewtype:=xlConditionValueNumber, NewValue:=0
```

Para que os primeiros 20 por cento das barras tenham a maior barra, utilize este código:

```
Range("A1:A10").FormatConditions(ThisCond).MaxPoint.ModifyNewtype:=xlConditionValuePercent, NewValue:=80
```

Uma alternativa interessante é mostrar apenas as barras de dados, e não o valor. Para fazer isso, utilize este código:

```
Range("A1:A10").FormatConditions(ThisCond).ShowValue = False
```

Para criar as barras de dados mostradas na coluna C da Figura 15.2, utilize este código:

```
Sub AddDataBar()
'Adiciona uma barra de dados.
'Assegura que quaisquer créditos < 0 tenham uma barra de dados como 0.
'
    Com Range("A1:A10").
        ' Adiciona as barras de dados.
        .FormatConditions.AddDataBar
        ' Configura o limite mais baixo.
        ThisCond = .FormatConditions.Count
        .FormatConditions(ThisCond).MinPoint.Modifynewtype:=xlConditionValueNumber, newvalue:=0
        ' Utiliza vermelho, mais escuro que o normal.
        With .FormatConditions(ThisCond).BarColor
            .Color = RGB(255, 0, 0)
            .TintAndShade = -0.5
        End With
    End With
End Sub
```

Adicionando escalas de cor a um intervalo

As escalas de cor podem ser adicionadas em variedades de escala de duas ou três cores. A Figura 15.3 mostra as configurações disponíveis na interface com o usuário do Excel para uma escala de cor que utiliza três cores.

Figura 15.3
As escalas de cor permitem mostrar pontos ativos no conjunto de dados.

Como a barra de dados, uma escala de cor é aplicada a um objeto do intervalo utilizando o método `AddColorScale`. Você deve especificar um `ColorScaleType` de `2` ou `3` como o único argumento do método `AddColorScale`.

Indique, então, uma cor e tinta para algum ou todos os critérios de escala de cor. Também é possível especificar se a sombra será aplicada ao valor mais baixo, ao valor mais alto, a um determinado valor, a uma percentagem ou a um percentil utilizando os valores mostrados anteriormente na Tabela 15.2.

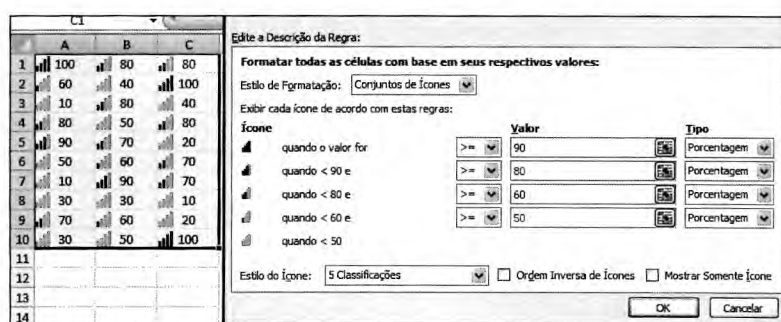
O próximo código gera uma escala de três cores no intervalo A1:A10:

```
Sub Add3ColorScale()
    With Range("A1:A10")
        .FormatConditions.Delete
        ' Adiciona a escala de cor como uma escala de 3 cores.
        .FormatConditions.AddColorScale ColorScaleType:=3

        ' Formata a primeira cor como vermelho-claro.
        .FormatConditions(1).ColorScaleCriteria(1).Type = xlConditionValuePercent
        .FormatConditions(1).ColorScaleCriteria(1).Value = 30
```

```
        .FormatConditions(1).ColorScaleCriteria(1).FormatColor.Color =RGB(255, 0, 0)
        .FormatConditions(1).ColorScaleCriteria(1).FormatColor.TintAndShade = 0.25

        ' Formata a segunda cor como verde a 50%.
        .FormatConditions(1).ColorScaleCriteria(2).Type = xlConditionValuePercent
        .FormatConditions(1).ColorScaleCriteria(2).Value = 50
        .FormatConditions(1).ColorScaleCriteria(2).FormatColor.Color =RGB(0, 255, 0)
        .FormatConditions(1).ColorScaleCriteria(2).FormatColor.TintAndShade = 0

        ' Formata a terceira cor como azul-escuro.
        .FormatConditions(1).ColorScaleCriteria(3).Type = xlConditionValuePercent
        .FormatConditions(1).ColorScaleCriteria(3).Value = 80
        .FormatConditions(1).ColorScaleCriteria(3).FormatColor.Color =RGB(0, 0, 255)
        .FormatConditions(1).ColorScaleCriteria(3).FormatColor.TintAndShade = -0.25
    End With

End Sub
```

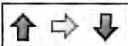
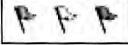
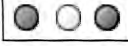
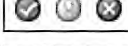
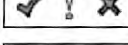
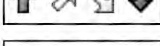
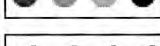
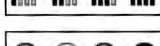
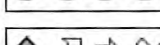
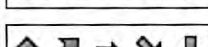
Adicionando conjunto de ícones a um intervalo

Os conjuntos de ícones no Excel vêm em três, quatro ou cinco diferentes ícones no conjunto. A Figura 15.4 mostra as configurações de um conjunto com cinco ícones diferentes.

Figura 15.4
Com ícones adicionais, a complexidade do código aumenta.

Para adicionar um conjunto de ícones a um intervalo, utilize o método `AddIconSet`. Nenhum argumento é necessário. Então, ajuste as três propriedades que se aplicam ao conjunto de ícones. Em seguida, utilize várias linhas de código adicionais para especificar o conjunto de ícones em uso e os limites para cada um.

Especificando um conjunto de ícones

Depois de adicionar o conjunto de ícones, você pode controlar se a ordem do ícone será invertida, se o Excel mostrará apenas os ícones e, então, especificar 1 dos 16 conjuntos de ícones predefinidos:

```
With Range("A1:C10")
    .FormatConditions.Delete
    .FormatConditions.AddIconSetCondition
    ' Configurações globais para o conjunto de ícones.
     With .FormatConditions(1)
        .ReverseOrder = False
        .ShowIconOnly = False
        .IconSet = ActiveWorkbook.IconSets(xl5CRV)
    End With
End With
```

> **NOTA** É um pouco curioso que a coleção IconSets seja uma propriedade da pasta de trabalho ativa. Isso parece indicar que, nas versões futuras do Excel, novos conjuntos de ícones podem estar disponíveis.

A Tabela 15.3 mostra a lista completa de conjuntos de ícones.

Tabela 15.3 Conjuntos de ícones disponíveis e suas constantes do VBA

Ícone	Valor	Descrição	Constante
	1	3 Setas	`xl3Arrows`
	2	3 Setas (Cinza)	`xl3ArrowsGray`
	3	3 Sinalizadores	`xl3Flags`
	4	3 Semáforos (Não-Coroados)	`xl3TrafficLights1`
	5	3 Semáforos (Coroados)	`xl3TrafficLights2`
	6	3 Sinais	`xl3Signs`
	7	3 Símbolos	`xl3Symbols`
	8	3 Símbolos (Circulados)	`xl3Symbols2`
	9	4 setas	`xl4Arrows`
	10	4 Setas (Cinza)	`xl4ArrowsGray`
	11	4 Vermelho para Preto	`xl4RedToBlack`
	12	4 Classificações	`xl4CRV`
	13	4 Semáforos (Coroados)	`xl4TrafficLights`
	14	5 Setas	`xl5Arrows`
	15	5 Setas (Cinza)	`xl5ArrowsGray`
	16	5 Classificações	`xl5CRV`
	17	5 Quartos	`xl5Quarters`

Especificando intervalos para cada ícone

Depois de especificar o tipo de conjunto de ícone, você pode especificar intervalos para cada ícone dentro do conjunto. Por padrão, o primeiro ícone inicia no valor mais baixo. Você pode ajustar as configurações para cada um dos ícones adicionais no conjunto:

```
With Range("A1:C10")
        ' O primeiro ícone sempre inicia em 0.
        ' As configurações para o segundo ícone iniciam em 50%.
        With .FormatConditions(1).IconCriteria(2)
            .Type = xlConditionValuePercent
            .Value = 50
            .Operator = xlGreaterEqual
        End With
        With .FormatConditions(1).IconCriteria(3)
            .Type = xlConditionValuePercent
            .Value = 60
            .Operator = xlGreaterEqual
        End With
        With .FormatConditions(1).IconCriteria(4)
            .Type = xlConditionValuePercent
            .Value = 80
```

```
            .Operator = xlGreaterEqual
        End With
        With .FormatConditions(1).IconCriteria(5)
            .Type = xlConditionValuePercent
            .Value = 90
            .Operator = xlGreaterEqual
        End With
    End With
```

Valores válidos para a propriedade `Operator` são `XlGreater` ou `xlGreaterEqual`.

ATENÇÃO

Com o VBA, é muito fácil criar intervalos sobrepostos (por exemplo, ícone 1 de 0 a 50, ícone 2 de 30 a 90). Embora a caixa de diálogo Editar Regra de Formatação impeça intervalos sobrepostos, o VBA permite. Seu conjunto de ícones será exibido de modo imprevisível se você criar intervalos inválidos.

Utilizando truques de visualização

Se você utilizar um conjunto de ícones ou uma escala de cor, o Excel aplicará uma cor a todas as células no conjunto de dados. Dois truques nesta seção permitem aplicar um conjunto de ícones a apenas um subconjunto das células ou aplicar duas barras de cor de dados diferentes ao mesmo intervalo. O primeiro truque está disponível na interface com o usuário, mas o segundo está disponível apenas no VBA.

Criando um conjunto de ícones a um subconjunto de um intervalo

Algumas vezes, você pode querer aplicar apenas um X vermelho às células problemáticas em um intervalo. É difícil fazer isso na interface com o usuário.

Na interface com o usuário, siga estes passos para aplicar um X vermelho aos valores maiores que 80:

1. Adicione um conjunto de ícones de três símbolos ao intervalo.
2. Especifique que os símbolos devem ser invertidos.
3. Indique que o terceiro ícone deverá aparecer para valores maiores que `80`. Você tem agora uma mistura dos três ícones, como mostrado na Figura 15.5.

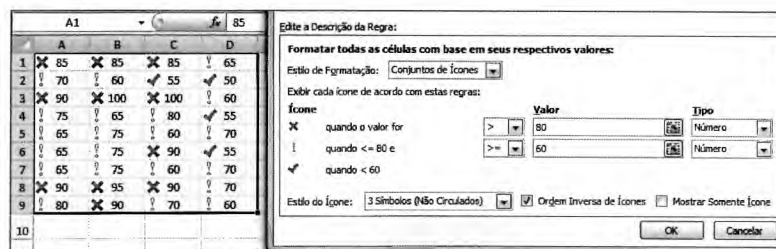
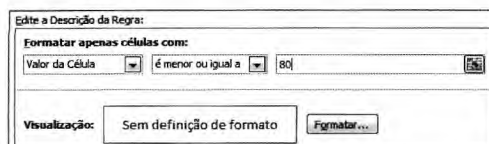
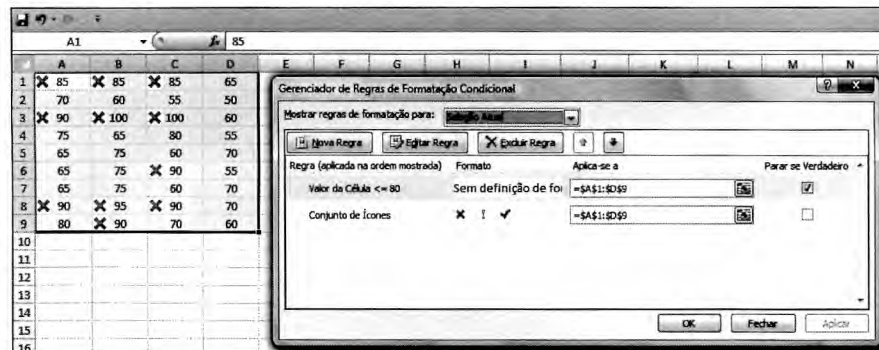
Figura 15.5 Primeiro adicione um conjunto de três ícones, prestando muita atenção ao valor do X vermelho.

4. Adicione uma novo formato condicional para realçar células menores que ou iguais a `80`. Não especifique nenhuma formatação especial às células que correspondem a essa regra, como mostrado na Figura 15.6.

Figura 15.6 A nova formatação condicional parece simplória — quando o valor for menor que ou igual a `80`, nada acontecerá.

O resultado é que apenas as células maiores que `80` serão exibidas com um X vermelho, como mostrado na Figura 15.7.

O código para criar esse efeito no VBA é relativamente simples e direto. Grande parte do código é usada para garantir que o conjunto de ícones tem os símbolos X vermelhos nas células maiores que `80`.

Quando você utiliza o método `FormatConditions.Add` para adicionar a segunda condição, o Excel refere-se inicialmente a essa condição como `FormatConditions(2)`. Mas você precisa assegurar que essa condição seja executada primeiro, para então utilizar o método `SetFirstPriority` para mover a nova condição para o início da lista. O passo final é desativar a propriedade `StopIfTrue`; mas você precisa saber que a nova condição é referida como `FormatConditions(1)` depois de executar o método `SetFirstPriority`.

Figura 15.7
Mas quando você pede que o Excel pare de avaliar as regras depois que a regra <=80 for true, o Excel nunca tem uma chance de adicionar a marca de seleção ou o ponto de exclamação às outras células.

O código para realçar valores maiores que 80 com um X vermelho é mostrado aqui:

```
Sub TrickyFormatting()
    With Range("A1:D9")
        .FormatConditions.Delete
        ' Adiciona e formata os 3 ícones de símbolos.
        .FormatConditions.AddIconSetCondition
        With .FormatConditions(1)
            .ReverseOrder = True
            .ShowIconOnly = False
            .IconSet = ActiveWorkbook.IconSets(xl3Symbols2)
        End With
        ' O limiar para esse ícone realmente não importa,
        ' mas você tem de certificar-se de que ele não sobrepõe o ícone 3.
        With .FormatConditions(1).IconCriteria(2)
            .Type = xlConditionValue
            .Value = 66
            .Operator = xlGreater
        End With
        ' Certifica-se de que o X vermelho aparece para células acima de 80.
        With .FormatConditions(1).IconCriteria(3)
            .Type = xlConditionValue
            .Value = 80
            .Operator = xlGreater
        End With

        ' Em seguida, adiciona uma condição para capturar itens < = 80.
        .FormatConditions.Add Type:=xlCellValue,Operator:=xlLessEqual, Formula1:="=80"
        ' Move essa nova condição da posição 2 para a posição 1.
        .FormatConditions(2).SetFirstPriority
        ' A nova condição é agora o índice nº1. Adiciona Parar se Verdadeiro.
        .FormatConditions(1).StopIfTrue = True
    End With
End Sub
```

Utilizando duas cores de barras de dados em um intervalo

Esse truque é particularmente interessante porque só pode ser obtido com o VBA. Digamos que valores acima de 90 são aceitáveis e abaixo de 90 indicam problemas. Você gostaria que os valores aceitáveis tivessem uma barra verde e os outros, uma barra vermelha.

Utilizando o VBA, você primeiramente adiciona as barras de dados verdes. Então, sem remover a condição de formato, adiciona as barras de dados vermelhas.

No VBA, todas as condições de formato têm uma propriedade Formula que define se uma condição será exibida para determinada célula. Então, o truque é escrever uma fórmula que defina quando as barras verdes serão exibidas. Quando a fórmula não for True, as barras vermelhas terão permissão de aparecer.

Na Figura 15.8, o efeito está sendo aplicado ao intervalo A1:D10. Você precisa escrever a fórmula no estilo A1, como se ele se aplicasse ao canto superior esquerdo da seleção. A fórmula tem de avaliar True ou False. O Excel automaticamente copia a fórmula em todas as células no intervalo. A fórmula para essa condição é =IF(A1>90,True,False).

DICA A fórmula é avaliada em relação à localização atual do ponteiro de célula. Embora normalmente não seja necessário selecionar células antes de adicionar um FormatCondition, nesse caso, selecionar o intervalo assegura o funcionamento da fórmula.

	A	B	C	D
1	92	96	81	88
2	88	84	82	99
3	99	85	92	88
4	84	84	82	84
5	90	90	82	99
6	90	80	98	88
7	81	97	81	85
8	89	89	91	93
9	81	94	88	83
10	87	82	86	85

Figura 15.8
As barras escuras são vermelhas e as mais claras, verdes. O VBA foi utilizado para criar duas barras de dados sobrepostas e, então, a propriedade `Formula` ocultou as barras superiores para células abaixo de `90`.

O código seguinte cria as barras de dados de duas cores:

```
Sub AddTwoDataBars()
    With Range("A1:D10")
        .Select ' A .Formula abaixo requer .Select aqui.
        .FormatConditions.Delete
        ' Adiciona uma barra de dados verde-clara.
        .FormatConditions.AddDataBar
        .FormatConditions(1).BarColor.Color = RGB(0, 255, 0)
        .FormatConditions(1).BarColor.TintAndShade = 0.25
        ' Adiciona uma barra de dados vermelha.
        .FormatConditions.AddDataBar
        .FormatConditions(2).BarColor.Color = RGB(255, 0, 0)
        ' Torna as barras verdes apenas.
        .FormatConditions(1).Formula = "=IF(A1>90,True,False)"
    End With
End Sub
```

A propriedade Formula funciona para todos os formatos condicionais. Isso permite criar algumas combinações relativamente inconvenientes de visualizações de dados.

Na Figura 15.9, cinco conjuntos diferentes de ícones são combinados em um único intervalo. Naturalmente, ninguém seria capaz de descobrir se um sinalizador vermelho é pior que uma seta para baixo cinza, mas essa capacidade oferece combinações interessantes para aqueles com pouca criatividade.

	A	B	C
1	1	23	12
2	17	3	14
3	4	19	5
4	7	11	26
5	21	2	10
6	20	15	13
7	16	6	28
8	25	24	27
9	18	9	22
10	29	8	30

Figura 15.9
O VBA criou essa mistura de cinco conjuntos de ícones diferentes em um único intervalo. A propriedade `Formula` no VBA é a chave para combinar conjuntos de ícones.

```
Sub AddCrazyIcons()
    With Range("A1:C10")
        .Select ' As linhas .Formula abaixo requerem .Select aqui.
        .FormatConditions.Delete

        ' Primeiro conjunto de ícones.
        .FormatConditions.AddIconSetCondition
        .FormatConditions(1).IconSet = ActiveWorkbook.IconSets(xl3Flags)
        .FormatConditions(1).Formula = "=IF(A1<5,TRUE,FALSE)"

        ' Próximo conjunto de ícones.
        .FormatConditions.AddIconSetCondition
        .FormatConditions(2).IconSet = ActiveWorkbook.IconSets(xl3ArrowsGray)
        .FormatConditions(2).Formula = "=IF(A1<12,TRUE,FALSE)"

        ' Próximo conjunto de ícones.
        .FormatConditions.AddIconSetCondition
        .FormatConditions(3).IconSet = ActiveWorkbook.IconSets(xl3Symbols2)
        .FormatConditions(3).Formula = "=IF(A1<22,TRUE,FALSE)"

        ' Próximo conjunto de ícones.
        .FormatConditions.AddIconSetCondition
```

```
        .FormatConditions(4).IconSet = ActiveWorkbook.IconSets(xl4CRV)
        .FormatConditions(4).Formula = "=IF(A1<27,TRUE,FALSE)"

        ' Próximo conjunto de ícones.
        .FormatConditions.AddIconSetCondition
        .FormatConditions(5).IconSet = ActiveWorkbook.IconSets(xl5CRV)
    End With
End Sub
```

Utilizando outros métodos de formatação condicional

Embora os conjuntos de ícones, barras de dados e escalas de cor ganhem mais atenção, há ainda muitos outros usos para a formatação condicional.

Os exemplos restantes deste capítulo mostram algumas das regras de formatação condicional anteriores e alguns novos métodos disponíveis.

Formatando células que estão acima ou abaixo da média

Utilize o método `AddAboveAverage` para formatar células que estiverem acima ou abaixo da média. Depois de adicionar a formatação condicional, especifique se a propriedade `AboveBelow` é `xlAboveAverage` ou `xlBelowAverage`.

As duas macros a seguir realçam células acima e abaixo da média:

```
Sub FormatAboveAverage()
    With Selection
        .FormatConditions.Delete
        .FormatConditions.AddAboveAverage
        .FormatConditions(1).AboveBelow = xlAboveAverage
        .FormatConditions(1).Interior.Color = RGB(255, 0, 0)
    End With
End Sub

Sub FormatBelowAverage()
    With Selection
        .FormatConditions.Delete
        .FormatConditions.AddAboveAverage
        .FormatConditions(1).AboveBelow = xlBelowAverage
        .FormatConditions(1).Interior.Color = RGB(255, 0, 0)
    End With
End Sub
```

Formatando células nos 10 Primeiros ou 5 Últimos

Quatro das escolhas no menu flutuante Regra de Primeiros/Últimos são controladas com o método `AddTop10`. Depois de adicionar a condição de formato, você precisa configurar três propriedades que controlam como a condição é calculada:

- **`TopBottom`** — Configure essa propriedade como `xlTop10Top` ou `xlTop10Bottom`.
- **`Value`** — Configure essa propriedade como `5` para o primeiro 5, `6` para o primeiro 6 e assim por diante.
- **`Percent`** — Configure essa propriedade como `False` se você quiser a opção 10 Primeiros Itens. Configure como `True` se quiser os Primeiros 10% dos itens.

O código seguinte destaca as primeiras e as últimas células:

```
Sub FormatTop10Items()
    With Selection
        .FormatConditions.Delete
        .FormatConditions.AddTop10
        .FormatConditions(1).TopBottom = xlTop10Top
        .FormatConditions(1).Value = 10
        .FormatConditions(1).Percent = False
        .FormatConditions(1).Interior.Color = RGB(255, 0, 0)
    End With
End Sub

Sub FormatBottom5Items()
    With Selection
        .FormatConditions.Delete
```

```
        .FormatConditions.AddTop10
        .FormatConditions(1).TopBottom = xlTop10Bottom
        .FormatConditions(1).Value = 5
        .FormatConditions(1).Percent = False
        .FormatConditions(1).Interior.Color = RGB(255, 0, 0)
    End With
End Sub

Sub FormatTop12Percent()
    With Selection
        .FormatConditions.Delete
        .FormatConditions.AddTop10
        .FormatConditions(1).TopBottom = xlTop10Top
        .FormatConditions(1).Value = 12
        .FormatConditions(1).Percent = True
        .FormatConditions(1).Interior.Color = RGB(255, 0, 0)
    End With
End Sub
```

Formatando uma única célula ou células duplicadas

O comando Remover Duplicatas na guia Dados da Faixa é um comando destrutivo. Talvez você queira marcar as duplicatas sem removê-las. Se quiser, o método `AddUniqueValues` marca as células duplicadas ou únicas.

Depois de chamar o método, configure a propriedade `DupeUnique` como `xlUnique` ou `xlDuplicate`.

Como discursei inflamadamente sobre isso em *Special Edition Using Microsoft Office Excel 2007*, realmente não gosto de quaisquer dessas opções. Como você pode ver na Figura 15.10, escolher valores duplicados, como na coluna A, marca ambas as células que contêm a duplicata. Por exemplo, tanto A2 como A8 são marcadas, quando na realidade apenas A8 é o valor duplicado.

Escolher valores únicos, como na coluna B, marca apenas as células que não têm uma duplicata. Isso deixa várias células desmarcadas. Por exemplo, nenhuma das células que contém 17 é marcada.

Como qualquer analista de dados sabe, a opção verdadeiramente útil teria sido marcar o primeiro valor único. Nesse estado ansioso, o Excel marcaria uma instância de cada valor único. Nesse caso, os 17 em E2 seriam marcados, mas todas células subseqüentes que contêm 17, como E8, permaneceriam desmarcadas.

Figura 15.10
O método `AddUniqueValues` pode marcar células, como na coluna A ou C. Infelizmente, esse método não pode marcar o padrão verdadeiramente útil na coluna E.

	A	B	C	D	E
1	Duplicado		Único		Desejável
2			17		
3			11		
4			7		
5			7		7
6			10		
7			10		10
8			17		17
9			11		11
10			14		
11			10		10
12	12				
13			14		14
14	2				
15	18				
16	4				

ATENÇÃO

Para obter informações úteis na coluna E, veja o código HighlightFirstUnique, na página 281.

O código para marcar valores duplicados ou únicos é mostrado aqui:

```
Sub FormatDuplicate()
    With Selection
        .FormatConditions.Delete
        .FormatConditions.AddUniqueValues
        .FormatConditions(1).DupeUnique = xlDuplicate
        .FormatConditions(1).Interior.Color = RGB(255, 0, 0)
    End With
End Sub

Sub FormatUnique()
```

```
    With Selection
        .FormatConditions.Delete
        .FormatConditions.AddUniqueValues
        .FormatConditions(1).DupeUnique = xlUnique
        .FormatConditions(1).Interior.Color = RGB(255, 0, 0)
    End With
End Sub
```

Formatando células com base em seu valor

Os formatos condicionais de valor já existem há várias versões do Excel. Utilize o método `Add` com os seguintes argumentos:

- **`Type`** — Nesta seção, o tipo será `xlCellValue`.
- **`Operator`** — Pode ser `xlBetween`, `xlEqual`, `xlGreater`, `xlGreaterEqual`, `xlLess`, `xlLessEqual`, `xlNotBetween`, `xlNotEqual`.
- **`Formula1`** — `Formula1` é utilizado com cada um dos operadores especificados para fornecer um valor numérico.
- **`Formula2`** — Esse argumento é utilizado para `xlBetween` e `xlNotBetween`.

A amostra de código seguinte destaca células com base em seus valores:

```
Sub FormatBetween10And20()
    With Selection
        .FormatConditions.Delete
        .FormatConditions.Add Type:=xlCellValue, Operator:=xlBetween,Formula1:="=10", Formula2:="=20"
        .FormatConditions(1).Interior.Color = RGB(255, 0, 0)
    End With
End Sub

Sub FormatLessThan15()
    With Selection
        .FormatConditions.Delete
        .FormatConditions.Add Type:=xlCellValue, Operator:=xlLess,Formula1:="=15"
        .FormatConditions(1).Interior.Color = RGB(255, 0, 0)
    End With
End Sub
```

Formatando células que contêm texto

Ao tentar destacar células que contêm um certo fragmento de texto, você utilizará o método `Add`, o tipo `xlTextString` e um operador de `xlBeginsWith`, `xlContains`, `xlDoesNotContain` ou `xlEndsWith`.

O código seguinte destaca todas as células que contêm uma letra maiúscula A:

```
Sub FormatContainsA()
    With Selection
        .FormatConditions.Delete
        .FormatConditions.Add Type:=xlTextString, String:="A",TextOperator:=xlContains
        ' Outras escolhas: xlBeginsWith, xlDoesNotContain, xlEndsWith.
        .FormatConditions(1).Interior.Color = RGB(255, 0, 0)
    End With
End Sub
```

Formatando células que contêm datas

Os formatos condicionais de data são novos no Excel 2007. A lista de operadores de data disponíveis é um subconjunto de operadores de data disponíveis nos novos filtros de tabela dinâmica. Utilize o método `Add`, o tipo `xlTimePeriod` e um desses valores `DateOperator`: `xlYesterday`, `xlToday`, `xlTomorrow`, `xlLastWeek`, `xlLast7Days`, `xlThisWeek`, `xlNextWeek`, `xlLastMonth`, `xlThisMonth`, `xlNextMonth`.

O seguinte código destaca todas as datas da semana passada:

```
Sub FormatDatesLastWeek()
    With Selection
        .FormatConditions.Delete
        ' As escolhas DateOperator incluem xlYesterday, xlToday, xlTomorrow,
        ' xlLastWeek, xlThisWeek, xlNextWeek, xlLast7Days
        ' xlLastMonth, xlThisMonth, xlNextMonth,
        .FormatConditions.Add Type:=xlTimePeriod, DateOperator:=xlLastWeek
        .FormatConditions(1).Interior.Color = RGB(255, 0, 0)
    End With
End Sub
```

Formatando células que contêm *lacunas* ou erros

Enterradas no fundo da interface do Excel estão as opções para formatar células que contêm lacunas, que contêm erros, que não contêm lacunas e que não contêm erros. Se você utilizar o programa de gravação de macros, o Excel aplicará a complicada versão `xlExpression` de formatação condicional. Por exemplo, para procurar por uma lacuna, o Excel testará se `=LEN(TRIM(A1))=0`. Em vez disso, você pode utilizar qualquer um desses quatro tipos auto-explicativos. Com eles, não será solicitado a utilizar nenhum outro argumento.

```
.FormatConditions.Add Type:=xlBlanksCondition
.FormatConditions.Add Type:=xlErrorsCondition
.FormatConditions.Add Type:=xlNoBlanksCondition
.FormatConditions.Add Type:=xlNoErrorsCondition
```

Utilizando uma fórmula para determinar que células formatar

O formato condicional mais poderoso ainda é o tipo `xlExpression`. Nesse tipo, você fornece uma fórmula para a célula ativa, que é avaliada como `True` ou `False`. Certifique-se de escrever a fórmula com referências relativas ou absolutas para que a fórmula esteja correta quando o Excel copiá-la nas células restantes na seleção.

Um número infinito de condições pode ser identificado com uma fórmula. Duas condições populares são mostradas aqui.

Destaque a primeira ocorrência única de cada valor em um intervalo

Na coluna A da Figura 15.11, vamos destacar a primeira ocorrência de cada valor na coluna. As células realçadas então conterão uma lista completa dos números únicos localizados na coluna.

A macro deve selecionar as células A1:A15. A fórmula deve ser escrita para retornar um valor `True` ou `False` para a célula A1. Como o Excel copia logicamente essa fórmula no intervalo inteiro, uma combinação cuidadosa de referências relativas e absolutas deve ser empregada.

A fórmula pode utilizar a função `CONT.SE`. Verifique quantas vezes o intervalo de A$1 a A1 contém o valor `A1`. Se o resultado for igual a `1`, a condição é `True` e a célula é destacada. A primeira fórmula é `=CONT.SE(A$1:A1;A1)=1`. Enquanto a fórmula é copiada em, digamos A12, a fórmula muda para `=CONT.SE(A$1:A12;A12)=1`.

	A	B	C	D	E	F
1				Região	Fatura	Vendas
2				Oeste	1001	112
3				Leste	1002	321
4	7			Central	1003	332
5				Oeste	1004	596
6	10			Leste	1005	642
7	17			Oeste	1006	700
8	11			Oeste	1007	253
9				Central	1008	529
10	10			Leste	1009	122
11				Oeste	1010	601
12	14			Central	1011	460
13						
14				Oeste	1013	763
15				Central	1014	193

Figura 15.11 Uma condição baseada em fórmula pode marcar a primeira ocorrência única de cada valor, como mostrado na coluna A, ou a linha inteira com as maiores vendas, como mostrado em D:F.

A próxima macro cria a formatação mostrada na coluna A da Figura 15.11:

```
Sub HighlightFirstUnique()
    With Range("A1:A15")
        .Select
        .FormatConditions.Delete
        .FormatConditions.Add Type:=xlExpression,Formula1:="=CONT.SE(A$1:A1;A1)=1"
        .FormatConditions(1).Interior.Color = RGB(255, 0, 0)
    End With
End Sub
```

Realce a linha inteira para o maior valor de vendas

Outro exemplo de condição baseada em fórmula é quando você quer realçar a linha inteira de um conjunto de dados em resposta a um valor em uma coluna. Considere o conjunto de dados nas células D2:F15 da Figura 15.11. Se você quiser realçar a linha inteira que contém a maior venda, selecione as células D2:F15 e escreva uma fórmula que funcione para a célula D2: `=$F2=MAX($F$2:$F$15)`. O código necessário para formatar a linha com o maior valor de vendas é o seguinte:

```
Sub HighlightWholeRow()
    With Range("D2:F15")
        .Select
        .FormatConditions.Delete
```

```
        .FormatConditions.Add Type:=xlExpression,Formula1:="=$F2=MAX($F$2:$F$15)"
        .FormatConditions(1).Interior.Color = RGB(255, 0, 0)
    End With
End Sub
```

Utilizando a nova propriedade `NumberFormat`

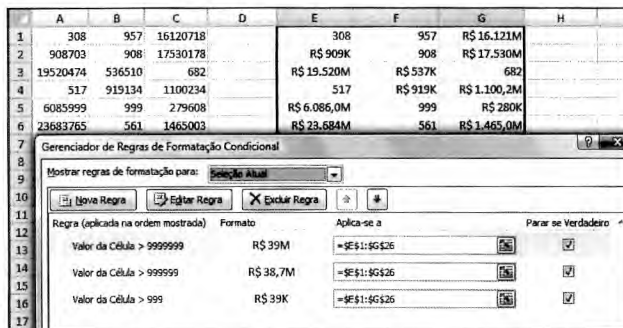
Nas versões anteriores do Excel, uma célula que correspondia a um formato condicional poderia ter determinada fonte, cor de fonte, borda ou padrão de preenchimento. No Excel 2007, também é possível especificar um formato de número. Isso pode ser útil para alterar seletivamente o formato de número utilizado para exibir os valores.

Por exemplo, talvez você queira exibir os números acima de 999 em milhares, números acima de 999.999 em centenas de milhares e números acima de 9 milhões em milhões.

Se você ativar o programa de gravação de macros e tentar gravar a configuração do formato condicional como um formato de número personalizado, o programa de gravação de macros VBA do Excel 2007 realmente gravará a ação de executar uma macro XL4! Pule o código gravado e utilize a propriedade `NumberFormat` como mostrado aqui:

```
Sub NumberFormat()
    With Range("E1:G26")
        .FormatConditions.Delete
        .FormatConditions.Add Type:=xlCellValue, Operator:=xlGreater,Formula1:="=9999999"
        .FormatConditions(1).NumberFormat = "$#,##0,""M"""
        .FormatConditions.Add Type:=xlCellValue, Operator:=xlGreater,
           Formula1:="=999999"
        .FormatConditions(2).NumberFormat = "$#,##0.0,""M"""
        .FormatConditions.Add Type:=xlCellValue, Operator:=xlGreater,
           Formula1:="=999"
        .FormatConditions(3).NumberFormat = "$#,##0,K"
    End With
End Sub
```

A Figura 15.12 mostra números originais em colunas A:C. Os resultados da execução da macro são mostrados nas colunas E:G. A caixa de diálogo mostra as regras de formato condicional resultantes.

Figura 15.12 Novos no Excel 2007, os formatos condicionais podem determinar um formato de número específico.

Próximos passos

No Capítulo 16, "Lendo e gravando na Web", você aprenderá a utilizar consultas Web para importar automaticamente dados da Internet para seus aplicativos do Excel.

Lendo e gravando na Web

16

A Internet tornou-se predominante e mudou nossas vidas. Da área de trabalho, milhões de respostas estão disponíveis na ponta dos dedos. Além disso, publicar um relatório na Web permite que milhões de pessoas acessem instantaneamente suas informações.

Este capítulo discute as maneiras automatizadas de extrair dados da Web de planilhas, utilizando consultas Web. Também mostra como salvar dados de sua planilha diretamente para a Web.

NESTE CAPÍTULO

Obtendo dados da Web

Alguém em uma mesa em qualquer lugar do mundo pode obter valores de ações atualizados até o último minuto para um portfólio. A Figura 16.1 apresenta uma página Web do Finance.Yahoo.com, que mostra cotações de ações atuais para um portfólio teórico. Obviamente, o pessoal do Yahoo! entende a importância das planilhas, pois oferece um link para que os dados sejam descarregados em uma planilha.

Em vez fazer manualmente o download de dados de um site Web todo dia e depois importá-los para o Excel, você pode usar o recurso Consulta da Web no Excel, que permite recuperar automaticamente os dados de uma página Web.

As consultas Web podem ser configuradas para atualizar os dados da Web diariamente ou, até mesmo, a cada minuto. Embora consultas Web fossem originalmente bastante difíceis de definir, a interface com o usuário do Excel agora inclui um navegador Web que pode ser utilizado para construí-las.

Criando uma consulta Web manualmente e atualizando com o VBA

A maneira mais fácil de começar a aprender como fazer consultas Web é criar sua primeira consulta manualmente. Em qualquer navegador Web, vá para um site e insira as configurações necessárias para exibir as informações de seu interesse. No caso da Figura 16.1, o URL para exibir esse portfólio é o seguinte:

```
http://finance.yahoo.com/q/cq?d=v1&s=KO%2c+MSFT%2c+GOOG%2c+WGO%2c+HOG%2c+F
```

Abra o Excel e localize uma área em branco na planilha. Na guia Dados da faixa, em Obter Dados Externos, escolha Da Web. O Excel mostrará a caixa de diálogo Nova Consulta à Web, exibindo a home page do Internet Explorer. Copie o URL anterior na caixa de texto Endereço e clique em Ir. Em um instante, a página Web desejada será exibida na caixa de diálogo. Observe que, além da página Web, há diversos quadrados amarelos com uma seta preta. Esses quadrados estão localizados no canto superior esquerdo de várias tabelas na página Web. Clique no quadrado que contém os dados que você quer importar para o Excel. Nesse caso, você quer as informações de portfólio. Como mostrado na Figura 16.2, clique na tabela de cotações. No momento em que você clicar, uma borda azul confirmará a tabela que será importada. Depois de clicar, a seta amarela transforma-se em uma marca de seleção verde.

Clique no botão Importar na caixa de diálogo Nova Consulta à Web. Clique em OK na caixa de diálogo Importar Dados. Em alguns segundos, você verá os dados ativos importados em um intervalo da planilha, como mostrado na Figura 16.3.

Figura 16.1
Uma abundância de dados quase em tempo real está disponível gratuitamente em sites Web em toda parte. O Finance.Yahoo.com oferece até mesmo um link para download do portfólio em uma planilha. As consultas Web oferecem algo muito mais surpreendente do que fazer download do arquivo manualmente todos os dias.

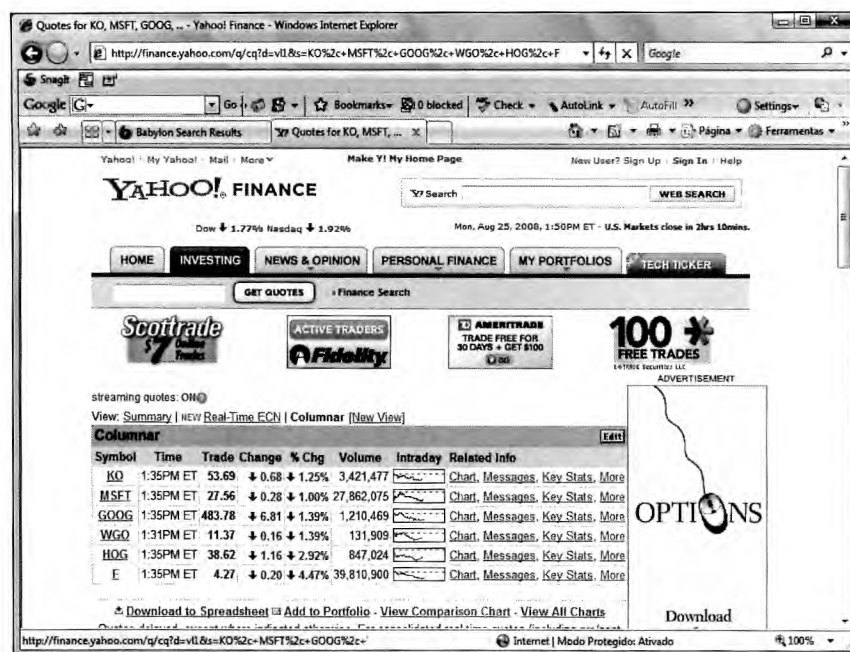



Figura 16.2
Utilize a caixa de diálogo Nova Consulta à Web para navegar a uma página Web. Realce a tabela que você deseja importar para o Excel clicando em uma seta amarela adjacente à tabela.

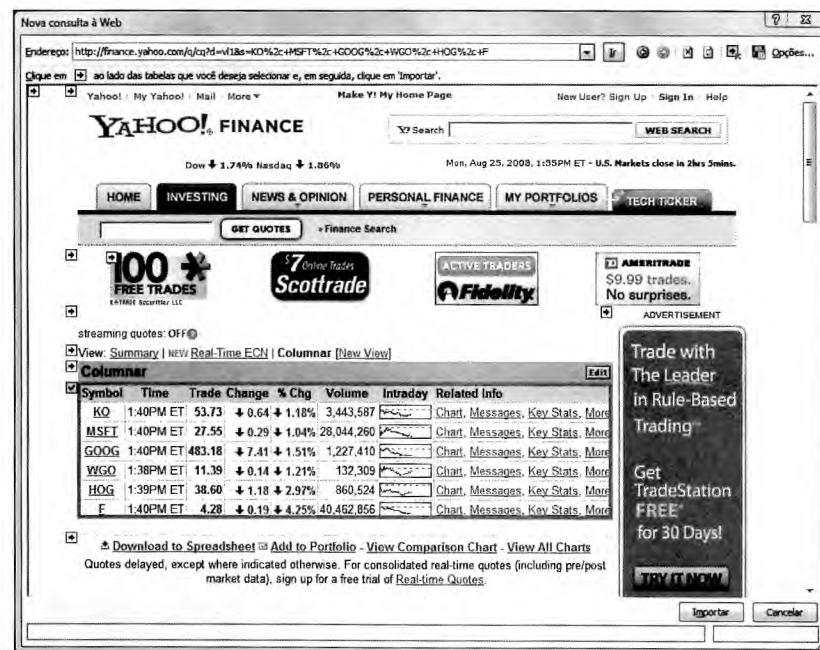
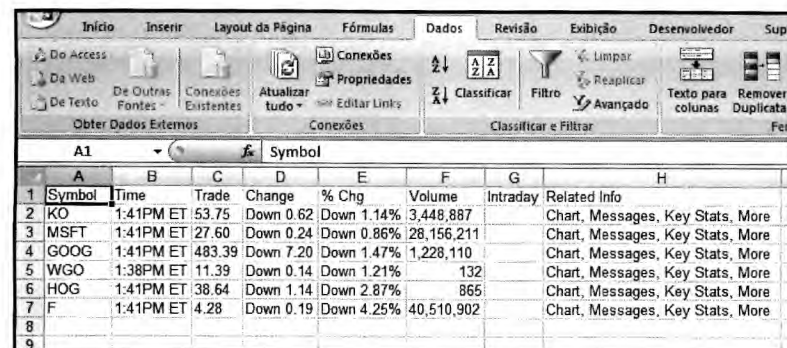



Figura 16.3
Os dados da página Web serão automaticamente copiados para sua planilha. Agora você pode utilizar o VBA para atualizar automaticamente esses dados ao seu comando ou periodicamente.

Utilizando o VBA para atualizar uma consulta Web existente

Para atualizar todas as consultas Web na planilha atual, utilize este código:

```
Sub RefreshAllWebQueries()
    Dim QT As QueryTable
    For Each QT In ActiveSheet.QueryTables
        Application.StatusBar = "Refreshing " & QT.Connection
        QT.Refresh
    Next QT
    Application.StatusBar = False
End Sub
```

Você pode atribuir essa macro a uma tecla de atalho (*hot key*) ou a um botão de macro e atualizar todas as consultas por demanda.

Construindo uma nova consulta Web com o VBA

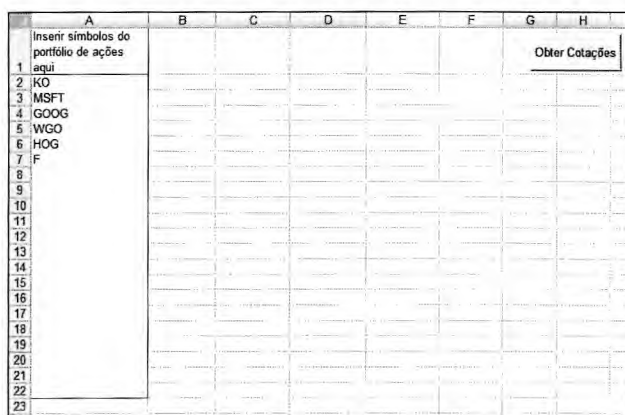
O problema com os exemplos anteriores é que o URL da consulta Web é codificado manualmente (*hard-coded*) no VBA. Alguém poderia ser solicitado a editar o código VBA toda vez que o portfólio mudasse.

É relativamente simples configurar uma consulta Web instantânea (*on-the-fly*). A chave é construir uma string de conexão. A string de conexão para a cotação de ações do site Finance Yahoo! é esta:

```
URL: http://finance.yahoo.com/q/cq?d=v1&s=PSO,+SJM,+KO,+MSFT,+CSCO,+INTC
```

Para configurar um aplicativo flexível, você precisa utilizar o caractere de concatenação (&) para unir a primeira parte da string de conexão aos vários símbolos de ações.

A Figura 16.4 mostra um *front-end* simples para um mecanismo de consulta Web. Qualquer pessoa pode inserir símbolos de ações de interesse na área sombreada da Coluna A. O botão Obter Cotações está configurado para chamar a macro CreateNewQuery.

Figura 16.4
Essa planilha serve como um front-end para permitir que diferentes símbolos de ações sejam inseridos na Coluna A. Clique no botão Obter Cotações para executar uma macro que cria uma consulta Web instantânea em uma segunda planilha e, depois, copia os dados para essa planilha.

Quando o botão é clicado, a macro cria uma string de conexão que concatena a primeira parte do URL com o primeiro símbolo de ações da célula A2:

```
ConnectString = "URL;http://finance.Yahoo.com/q/cq?d=v1&s=" &WSD.Cells(i, 1).Value
```

À medida que programa faz loop pelos símbolos de ações adicionais, ele aceita a string de conexão existente e adiciona uma vírgula e um sinal de adição e, depois, o próximo símbolo de ações:

```
ConnectString = ConnectString & ",+" & WSD.Cells(i, 1).Value
```

DICA Outra tarefa difícil é determinar o número da tabela que você quer recuperar da página Web. A solução mais fácil é registrar uma macro ao selecionar a tabela adequada. Em seguida, examine a macro gravada da propriedade WebTables = "11".

Depois que a string de conexão tiver sido construída, a macro utilizará uma planilha chamada Workspace para criar a consulta Web. Isso ocultará os resultados da consulta Web não formatados na visualização. Como a lista de símbolos de ações pode ter mudado desde a última consulta Web, o programa excluirá qualquer consulta antiga e, então, irá configurar a nova consulta.

É importante definir a consulta com `BackgroundRefresh` configurada como `False`. Isso assegura que a macro pause para dar tempo de a consulta se atualizar antes de continuar.

Depois que os resultados tiverem sido recebidos, o programa atribuirá um nome de intervalo aos resultados e utilizará fórmulas `PROCV` para recuperar as colunas desejadas da consulta Web:

```
Sub CreateNewQuery()
    Dim WSD As Worksheet
    Dim WSW As Worksheet
    Dim QT As QueryTable
    Dim FinalRow As Long
    Dim i As Integer
    Dim ConnectString As String
    Dim FinalResultRow As Long
    Dim RowCount As Long

    Set WSD = Worksheets("Portfolio")
    Set WSW = Worksheets("Workspace")

    ' Lê coluna A de Portfólio para localizar todos os símbolos de ações
    FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
    For i = 2 To FinalRow
        Select Case i
            Case 2
                ConnectString = "URL;http://finance.Yahoo.com/q/cq?d=v1&s=" &WSD.Cells(i, 1).Value
            Case Else
                ConnectString = ConnectString & "%2c+" & WSD.Cells(i, 1).Value
        End Select
    Next i

    ' Na planilha Workspace, limpa todas as tabelas de consulta existentes
    For Each QT In WSW.QueryTables
        QT.Delete
    Next QT

    ' Define uma nova consulta Web
    Set QT = WSW.QueryTables.Add(Connection:=ConnectString,Destination:=WSW.Range("A1"))
    With QT
        .Name = "portfolio"
        .FieldNames = True
        .RowNumbers = False
        .FillAdjacentFormulas = False
        .PreserveFormatting = True
        .RefreshOnFileOpen = False
        .BackgroundQuery = False
        .RefreshStyle = xlInsertDeleteCells
        .SavePassword = False
        .SaveData = True
        .AdjustColumnWidth = True
        .RefreshPeriod = 0
        .WebSelectionType = xlSpecifiedTables
        .WebFormatting = xlWebFormattingNone
        .WebTables = "11"
        .WebPreFormattedTextToColumns = True
        .WebConsecutiveDelimitersAsOne = True
        .WebSingleBlockTextImport = False
        .WebDisableDateRecognition = False
        .WebDisableRedirections = False
    End With

    ' Atualiza a consulta
    QT.Refresh BackgroundQuery:=False

    ' Define um intervalo nomeado para os resultados
    FinalResultRow = WSW.Cells(Rows,Count, 1).End(xlUp).Row
    WSW.Cells(1, 1).Resize(FinalResultRow, 7).Name = "WebInfo"

    ' Cria um PROCV para obter cotações de WSW para WSD
    RowCount = FinalRow - 1
    WSD.Cells(2, 2).Resize(RowCount, 1).FormulaR1C1 ="=VLOOKUP(RC1,WebInfo,3,False)"
    WSD.Cells(2, 3).Resize(RowCount, 1).FormulaR1C1 ="=VLOOKUP(RC1,WebInfo,4,False)"
    WSD.Cells(2, 4).Resize(RowCount, 1).FormulaR1C1 ="=VLOOKUP(RC1,WebInfo,5,False)"
    WSD.Cells(2, 5).Resize(RowCount, 1).FormulaR1C1 ="=VLOOKUP(RC1,WebInfo,6,False)"
    WSD.Cells(2, 6).Resize(RowCount, 1).FormulaR1C1 ="=VLOOKUP(RC1,WebInfo,2,False)"

    MsgBox "Os dados foram atualizados"
End Sub
```

Quando o programa executar, os resultados estarão bem-formatados, como mostrado na Figura 16.5.

	A	B	C	D	E	F	G	H
1	Inserir símbolos do portfólio de ações aqui						Obter Cotações	
2	KO	53.52	Down 0.85	Down 1.56%	3,799,091	02:30:45		
3	MSFT	27.49	Down 0.35	Down 1.26%	33,259,825	03:30:45		
4	GOOG	482.30	Down 8.29	Down 1.69%	1,376,061	04:30:45		
5	WGO	11.33	Down 0.20	Down 1.73%	177,009	05:30:45		
6	HOG	386.645	Down 1.1155	Down 2.80%	1,039,147	06:30:45		
7	F	4.33	Down 0.14	Down 3.13%	53,154,789	07:30:45		
8								
9								
10								
11								

Figura 16.5
Depois que você executar a macro, apenas as colunas relevantes serão mostradas no relatório. A planilha por trás contém a consulta Web não formatada do site Web.

Utilizando dados de streaming

Diversos serviços oferecem dados streaming ao vivo em sua planilha. Esses serviços utilizam uma tecnologia conhecida como troca de dados dinâmica (*dynamic data exchange* – DDE) para direcionar informações diretamente para as células da sua planilha de maneira automática. Embora esses serviços exijam uma assinatura mensal, é surpreendente observar a planilha mudar automaticamente a cada segundo com os preços das ações em tempo real.

Esses serviços DDE em tempo real em geral utilizam uma fórmula que identifica o programa EXE externo, um caractere | (barra vertical) e dados para o programa externo utilizar.

Na Figura 16.6, a fórmula na célula A2 está chamando o programa MktLink.exe para retornar dados de preços para o símbolo de ações AA.

A2 =Mktlink|Price!"[H1]AA n;3'

	A	B	C	D	E
1	AA	ABT	ABX	ACE	
2	37.77	52.26	14.58	34.85	
3					

Figura 16.6
O serviço MktLink.exe permite que os dados streaming ao vivo sejam direcionados para células na planilha.

Embora seja incrível assistir a esses serviços, os dados se movem muito rapidamente. O que fazer com informações que se atualizam em poucos segundos? O poder real é valer-se do Excel para capturar e salvar os dados de vez em quando para procurar tendências.

Utilizando `Application.OnTime` para analisar dados periodicamente

O VBA oferece o método `OnTime` para executar qualquer procedimento VBA em uma hora específica do dia ou depois que um determinado horário tiver passado.

Você poderia escrever uma macro que capturasse dados por hora durante todo o dia. Essa macro teria as horas codificadas manualmente. Teoricamente, o código a seguir irá capturar dados de um site Web de hora em hora durante o dia:

```
Sub ScheduleTheDay()
    Application.OnTime EarliestTime:=TimeValue("8:00 AM"),Procedure:=CaptureData
    Application.OnTime EarliestTime:=TimeValue("9:00 AM"),Procedure:=CaptureData
    Application.OnTime EarliestTime:=TimeValue("10:00 AM"),Procedure:=CaptureData
    Application.OnTime EarliestTime:=TimeValue("11:00 AM"),Procedure:=CaptureData
    Application.OnTime EarliestTime:=TimeValue("12:00 AM"),Procedure:=CaptureData
    Application.OnTime EarliestTime:=TimeValue("1:00 PM"),Procedure:=CaptureData
    Application.OnTime EarliestTime:=TimeValue("2:00 PM"),Procedure:=CaptureData
    Application.OnTime EarliestTime:=TimeValue("3:00 PM"),Procedure:=CaptureData
    Application.OnTime EarliestTime:=TimeValue("4:00 PM"),Procedure:=CaptureData
    Application.OnTime EarliestTime:=TimeValue("5:00 PM"),Procedure:=CaptureData
End Sub

Sub CaptureData()
    Dim WSQ As Worksheet
    Dim NextRow As Long
    Set WSQ = Worksheets("MyQuery")
    ' Atualiza a consulta Web
    WSQ.Range("A2").QueryTable.Refresh BackgroundQuery:=False
    ' Certifica-se de que os dados são atualizados
    Application.Wait (Now + TimeValue("0:00:10"))
    ' Copia os resultados da consulta Web para uma nova linha
    NextRow = WSQ.Cells(65536, 1).End(xlUp).Row + 1
    WSQ.Range("A2:B2").Copy WSQ.Cells(NextRow, 1)
End Sub
```

Os procedimentos agendados requerem o modo Pronto

O método `OnTime` executará contanto que apenas o Excel esteja no modo Pronto, Copiar, Recortar ou Localizar na hora definida. Se você começar a editar uma célula às 7:59:55 da manhã e mantiver essa célula no modo de edição, o Excel não poderá executar a macro CaptureData às 8:00 da manhã, conforme direcionado.

No exemplo anterior de código, especifiquei apenas a hora inicial para o procedimento executar. O Excel espera ansiosamente até que a planilha volte para o modo Pronto e, então, executa o programa agendado assim que puder.

O exemplo clássico é aquele em que você começa a editar uma célula às 7:59 da manhã e, então, seu gerente entra na sala e pede-lhe que participe de uma reunião do departamento. Se você deixar a planilha no modo de edição e participar da reunião até às 10:30 da manhã, o programa não poderá executar as primeiras três horas agendadas de atualizações. Assim que você voltar para sua mesa e pressionar Enter para sair do modo de edição, o programa executará todas as tarefas anteriormente agendadas. No código anterior, você verá que as três primeiras atualizações agendadas do programa aconteceram todas entre 10:30 e 10:31 da manhã.

Especificando uma janela Tempo para uma atualização

Uma alternativa é fornecer ao Excel uma janela de tempo na qual fazer a atualização. O código a seguir solicita ao Excel que execute a atualização a qualquer hora entre 8:00 e 8:05 da manhã. Se a sessão Excel permanecer no modo de edição por cinco minutos inteiros, a tarefa agendada será pulada:

```
Application.OnTime EarliestTime:=TimeValue("8:00 AM"), Procedure:=CaptureData,_
    LatestTime:=TimeValue("8:05 AM")
```

Cancelando uma macro agendada anteriormente

É relativamente difícil cancelar uma macro agendada anteriormente. É preciso saber a hora exata do agendamento de execução da macro. Para cancelar uma operação pendente, chame o método `OnTime` novamente, utilizando o parâmetro `Schedule:=False` para desfazer o agendamento do evento. O código seguinte cancela a execução das 11:00 da manhã de CaptureData:

```
Sub CancelEleven()
Application.OnTime EarliestTime:=TimeValue("11:00 AM"),Procedure:=CaptureData, Schedule:=False
End Sub
```

É interessante notar que as agendas OnTime são lembradas por uma instância em execução do Excel. Se você deixar o Excel aberto, mas fechar a pasta de trabalho com o procedimento agendado, ele ainda executará. Considere essa série hipotética de eventos:

1. Abra o Excel às 7:30 da manhã.
2. Abra `Schedule.XLS` e execute uma macro para agendar um procedimento às 8:00 da manhã.
3. Feche Schedule.xls, mas mantenha o Excel aberto.
4. Abra uma nova pasta de trabalho e comece a inserir os dados.

Às 8:00 da manhã, o Excel reabre Schedule.xls e executa a macro agendada. O Excel não fecha Schedule.xls. Como você pode imaginar, é muito irritante e preocupante quando não se está esperando por isso. Se você for fazer um uso extenso de `Application.Ontime`, talvez queira executá-la em uma instância do Excel ao trabalhar em uma segunda instância do programa.

Se estiver utilizando uma macro para agendar uma macro horas mais tarde a partir da hora atual, você pode lembrar-se da hora em uma célula separada e isolada para poder cancelar a atualização. Veja um exemplo na seção "Agendando uma macro para executar x minutos no futuro".

Fechar o Excel cancela todas as macros agendadas pendentes

Se você *fechar* o Excel com Arquivo, Sair, todas as macros agendadas serão automaticamente canceladas. Quando tiver uma macro que tenha agendado algumas macros em horários indeterminados, fechar o Excel é a única maneira de impedir a execução delas.

Agendando uma macro para executar x minutos no futuro

Você pode agendar a execução de uma macro por vez em um horário. A macro utiliza a função `TIME` para retornar a hora atual e adicionar 2 minutos e 30 segundos ao horário. A macro seguinte executa alguma coisa daqui a 2 minutos e 30 segundos:

```
Sub ScheduleAnything()
    ' Essa macro pode ser utilizada para agendar qualquer coisa
    WaitHours = 0
    WaitMin = 2
```

```
    WaitSec = 30
    NameOfScheduledProc = "CaptureData"
    ' --- Fim da Seção de Entrada ------

    ' Determina o próximo horário em que isso deve executar
    NextTime = Time + TimeSerial(WaitHours, WaitMin, WaitSec)

    ' Agenda ThisProcedure para executar depois
    Application.OnTime EarliestTime:=NextTime, Procedure:=NameOfScheduledProc

End Sub
```

Se você precisasse cancelar mais tarde esse evento agendado, seria quase impossível. Você não saberia a hora exata que a macro tivesse chegado à função TIME. Você pode tentar salvar esse valor em uma célula separada:

```
Sub ScheduleWithCancelOption
    NameOfScheduledProc = "CaptureData"

    ' Determina o próximo horário em que isso deve executar
    NextTime = Time + TimeSerial(0,2,30)
    Range("ZZ1").Value = NextTime

    ' Agenda ThisProcedure para executar depois
    Application.OnTime EarliestTime:=NextTime, Procedure:=NameOfScheduledProc

End Sub

Sub CancelLater()
        NextTime = Range("ZZ1").value
        Application.OnTime EarliestTime:=NextTime,Procedure:=CaptureData, Schedule:=False
End Sub
```

Agendando um lembrete verbal

O texto para ferramentas de fala no Excel pode ser divertido. A próxima macro configura uma agenda que o lembrará do horário da reunião do departamento:

```
Sub ScheduleSpeak()
    Application.OnTime EarliestTime:=TimeValue("9:14 AM"),Procedure:="RemindMe"
End Sub

Sub RemindMe()
    Application.Speech.Speak Text:="Bill. Está na hora da reunião do departamento."
End Sub
```

Se quiser pregar uma peça em um gerente, você pode agendar o Excel para ativar automaticamente o recurso Falar ao Entrar. Siga este roteiro:

1. Diga ao seu gerente que você o está levando para almoçar fora para comemorar o 1º de abril.
2. A determinada hora da manhã, enquanto seu gerente estiver tomando café, execute a macro ScheduleSpeech. Projete-a para executar 15 minutos depois da hora do almoço.
3. Saia com o gerente para almoçar.
4. Enquanto ele estiver longe, a macro agendada executará.
5. Quando ele voltar e começar a digitar os dados no Excel, o computador repetirá as células assim que elas forem inseridas. Isso lembra um pouco o computador, em *Jornadas nas Estrelas*, que repetia tudo que a tenente Uhura dizia.

Depois disso, você pode fingir-se inocente; afinal de contas, você tinha um forte álibi para pregar a peça nesse dia:

```
Sub ScheduleSpeech()
    Application.OnTime EarliestTime:=TimeValue("12:15 PM"),Procedure:="SetUpSpeech"
End Sub

Sub SetupSpeech())
    Application.Speech.SpeakCellOnEnter = True
End Sub
```

NOTA Para desativar o recurso Falar ao Entrar, pressione o botão do painel Personalização de QAT (veja na categoria chamada Comandos Que Não Estão na Faixa). Ou, se puder executar um VBA, altere a macro SetupSpeech para mudar de `True` para `False`.

Agendando a execução de uma macro a cada dois minutos

Meu método favorito é pedir que o Excel execute uma determinada macro a cada dois minutos. Mas percebi que, se uma macro apresentar um retardo porque deixei acidentalmente a pasta de trabalho no modo de edição enquanto estava na reunião do departamento, não quero que aconteça uma dezena de atualizações em questão de segundos.

A solução fácil é fazer o procedimento `ScheduleAnything` agendar-se recursivamente para executar outra vez em dois minutos. O próximo código agenda uma execução em dois minutos e, depois, executa CaptureData:

```
Sub ScheduleAnything()
    ' Essa macro pode ser utilizada para agendar qualquer coisa
    ' Insere a freqüência com que você quer executar a macro em horas e minutos
    WaitHours = 0
    WaitMin = 2
    WaitSec = 0
    NameOfThisProcedure = "ScheduleAnything"
    NameOfScheduledProc = "CaptureData"
    ' --- Fim da Seção de Entrada ------

    ' Determina o próximo horário em que isso deve executar
    NextTime = Time + TimeSerial(WaitHours, WaitMin, WaitSec)

    ' Agenda ThisProcedure para executar depois
    Application.OnTime EarliestTime:=NextTime, Procedure:=NameOfThisProcedure

    ' Obtém os dados
    Application.Run NameOfScheduledProc

End Sub
```

Esse método tem algumas vantagens. Não agendei um milhão de atualizações para o futuro. Tenho apenas uma atualização futura agendada para um determinado horário. Portanto, se acredito que estou cansado de assistir a dívida nacional a cada 15 segundos, só preciso comentar a linha de código Application.OnTime e esperar 15 segundos para que ocorra a última atualização.

Publicando dados em uma página Web

Este capítulo destacou muitas maneiras de capturar dados da Web. Ele também é útil para publicar dados do Excel na Web.

No Capítulo 14, "O poder do Excel", uma macro podia criar relatórios corporativos por regiões. Em vez de imprimir e enviar o relatório via fax, seria excelente poder salvar o arquivo Excel como HTML e postar os resultados em uma intranet corporativa para que o gerente regional pudesse acessar instantaneamente a última versão do relatório.

Considere um relatório semelhante ao mostrado na Figura 16.7. Com a interface com o usuário do Excel, é fácil salvar o relatório como uma página Web para criar uma visualização HTML dos dados.

	A	B	C
1	Os 5 maiores clientes na Região Oeste		
2			
3	Cliente	Receita	
4	Guarded Kettle Corporation	8.889K	
5	Agile Glass Supply	3.877K	
6	Tremendous Flagpole Traders	2.361K	
7	Innovative Oven Corporation	2.359K	
8	Trouble-Free Eggbeater Inc.	2.303K	
9	Total dos 5 maiores	19.789K	
10			

Figura 16.7 Uma macro do Capítulo 13 foi utilizada para gerar essa pasta de trabalho Excel automaticamente. Em vez de enviar o relatório via e-mail, poderíamos salvá-lo como página Web e postá-lo na intranet da empresa.

No Excel 2007, utilize Salvar Como, Outros Formatos, no menu Ícone do Office. Escolha Página Web (*.htm, *html) na lista suspensa Salvar como Tipo.

NOTA A opção do Excel 2003 para adicionar interatividade a uma página Web tornou-se obsoleta a partir do Excel 2007 e não está mais disponível.

Depois que a Microsoft removeu a opção de interatividade, você só precisa controlar o título que aparece na parte superior da janela. Clique no botão Alterar Título para alterar o tag `<Title>` da página Web. Digite um nome que acaba em .html ou .html e clique em Publicar...

O resultado é um arquivo que pode ser visualizado em qualquer navegador Web. A página Web mostra exatamente nossos formatos de números e tamanhos de fonte (veja Figura 16.8).

Várias versões atrás, a Microsoft apaixonou-se por salvar uma planilha como HTML e abrir o arquivo HTML no Excel e, então, deixar as fórmulas originais intactas. A Microsoft chamou esse recurso de *'round tripping'*. Esse recurso faz com que o Excel escreva arquivos HTML incrivelmente grandes. Os dados apresentados na Figura 16.9 referem-se a 228 bytes de dados, mas o Excel requer 5.662 bytes de dados para armazenar tanto os dados para a apresentação como os dados necessários para carregar o arquivo novamente no Excel.

Embora a apresentação dos dados da Figura 16.9 seja feita com exatidão, ela não é muito sofisticada. Não há o logotipo de empresa nem a barra de navegação para examinar outras informações.

Figura 16.8
Utilize o botão Alterar Título para adicionar um novo título à barra de título azul na parte superior da página Web.

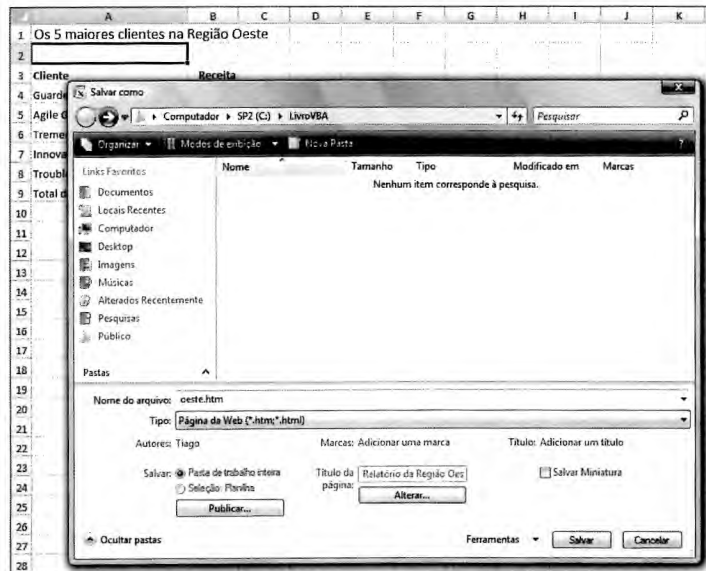
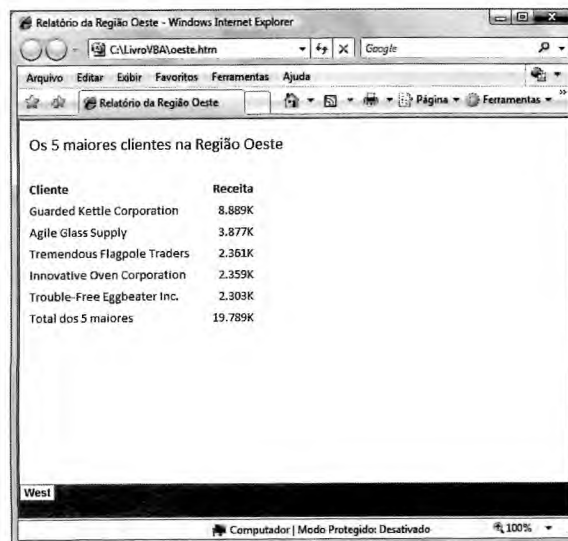



Figura 16.9
O Excel faz um bom trabalho de exibição da planilha como HTML.

Utilizando o VBA para criar páginas Web personalizadas

Muito antes de a Microsoft introduzir a funcionalidade Salvar Como Página Web, as pessoas utilizavam o VBA para obter dados do Excel e publicá-los como HTML. A vantagem desse método é que ele pode escrever instruções HTML específicas para exibir logotipos de empresa e barras de navegação.

Considere um típico modelo de página Web. Há código para exibir um logotipo e barra de navegação na parte superior lateral. Há dados específicos à página e dados para fechar o arquivo HTML. Construa um modelo HTML com as palavras **PUT DATA HERE** e examine a HTML resultante no Bloco de Notas, como mostrado na Figura 16.10.

Figura 16.10
Examinar o arquivo `sample.html` no Bloco de Notas e localizar as palavras *PUT DATA HERE* permite que o arquivo HTML seja separado em três partes: o início do código utilizado para escrever a barra de navegação, os dados e a última parte, utilizada para fechar a página HTML.

Depois de examinar o código no Bloco de Notas, a tarefa de criar uma página Web personalizada é relativamente simples. Primeiro, escreva o HTML necessário para desenhar as barras de navegação. Então, escreva os dados no Excel. Em seguida, escreva algumas linhas de HTML para fechar a tabela e terminar a página Web.

Utilizando o Excel como um sistema de gerenciamento de conteúdo

Duzentos e cinqüenta milhões de pessoas são proficientes no Excel. Em todo lugar, empresas usam o Excel e muitos funcionários se sentem à vontade para manter dados nesse programa. Em vez de forçar essas pessoas a aprender a criar páginas HTML, por que não construir um sistema de gerenciamento de conteúdo para pegar os dados do Excel e escrever páginas Web personalizadas?

A Figura 16.11 mostra um típico banco de dados de associados de uma organização. Um Web designer nos propôs a construção de um caro banco de dados PHP (Hypertext Preprocessor) na Web para exibir o diretório de associados e permitir a manutenção de membros.

A organização já está mantendo esses dados no Excel. Utilizando o código da Figura 16.10, você pode muito bem conhecer as partes inicial e final do código HTML necessário para exibir a página Web.

É simples criar um sistema de gerenciamento de conteúdo com essas ferramentas. Ao arquivo Excel do diretório de associados, adicionei duas planilhas. Na planilha chamada Parte Superior, copiei o HTML necessário para gerar a barra de navegação do site Web. Para a planilha chamada Parte Inferior, copiei o HTML necessário para gerar o fim da página HTML. A Figura 16.12 mostra a planilha Parte Inferior simples.

Figura 16.11
Em toda parte, empresas mantêm todos os tipos de dados no Excel e se sentem à vontade para atualizá-los nesse programa. Por que não combinar o Excel com um pouco de VBA *para criar código* HTML personalizado no Excel?

	A	B	C	D	E	F	G	H
1	empres	endere	cidade	estado	zip	fone	formatado	
2	ACCom	2930 Edi	Uniontow	OH	44685	3305551212	(330)555-1212	
3	ACTIO	11806 Fa	Uniontow	OH	44685	3305551212	(330)555-1212	
4	Allen Kr	2735 Ge	North	OH	44720-142	3305551212	(330)555-1212	
5	Allen M	3515 Me	Uniontow	OH	44685	3305551212	(330)555-1212	
6	Theresa	3500 Me	Uniontow	OH	44685	3305551212	(330)555-1212	
7	AlterCa	1420 Sm	Hertville	OH	44632	3305551212	(330)555-1212	
8	America	317 Sout	Hartville	OH	44632	3305551212	(330)555-1212	
9	America	2213 Cle	Canton	OH	44709	3305551212	(330)555-1212	
10	Arnold	504 W.	Hartville	OH	44632	3305551212	(330)555-1212	
11	ASAP,	6551 Mir	Canton	OH	44721	3305551212	(330)555-1212	
12	Aultman	2600	Canton	OH	44710	3305551212	(330)555-1212	
13	Baby Ty	878 Wes	Hertville	OH	44632	3305551212	(330)555-1212	
14	Best Bih	311 S. Pi	Hartville	OH	44632	3305551212	(330)555-1212	
15	Bicycle	854-A V	Hertville	OH	44632	3305551212	(330)555-1212	
16	Capital	150 Gen	Hartville	OH	44632	3305551212	(330)555-1212	
17	China	808 Wes	Hertville	OH	44632	3305551212	(330)555-1212	
18	ComDo	3458 Me	Uniontow	OH	44685	3305551212	(330)555-1212	
19	Compu	13163 M	Hertville	OH	44632	3305551212	(330)555-1212	
20	Concord	850 We.	Hartville	OH	44632	3305551212	(330)555-1212	
21	Cooper	300 N. C	Akron	OH	44333	3305551212	(330)555-1212	
22	Country	10244M	Hartville	OH	44632	3305551212	(330)555-1212	
23	Cutty's	8050 Edi	Louisville	OH	44641	3305551212	(330)555-1212	
24	Doreen'	831 Sum	Hertville	OH	44632	3305551212	(330)555-1212	

	A	B	C
1	Seqüência	Conteúdo	
2	1		
3	2	</td>	
4	3	</tr>	
5	4	</table>	
6	5	</BODY>	
7	6	</HTML>	
8			

Figura 16.12
Duas planilhas são adicionadas à pasta de trabalho do diretório de associados. Uma delas lista o código HTML necessário para desenhar a barra de navegação. A outra lista o código HTML necessário para concluir a página Web depois de exibir os dados.

O código da macro abre um arquivo de texto chamado `directory.html` para a saída. Primeiro, todo o código HTML da planilha chamada Parte Superior é gravado no arquivo.

Então, a macro faz loop por cada linha no diretório de associados, gravando os dados no arquivo.

Depois de completar esse loop, a macro escreve o código HTML da planilha Parte Inferior para terminar o arquivo:

```
Sub WriteMembershipHTML()
    ' Escreve páginas Web
    Dim WST As Worksheet
    Dim WSB As Worksheet
    Dim WSM As Worksheet
    Set WSB = Worksheets("Parte Inferior")
    Set WST = Worksheets("Parte Superior")
    Set WSM = Worksheets("Membros")

    ' Descobre o caminho
    MyPath = ThisWorkbook.Path

    LineCtr = 0

    FinalT = WST.Cells(Rows.Count, 1).End(xlUp).Row
    FinalB = WSB.Cells(Rows.Count, 1).End(xlUp).Row
    FinalM = WSM.Cells(Rows.Count, 1).End(xlUp).Row

    MyFile = "sampledirectory.html"

    ThisFile = MyPath & Application.PathSeparator & MyFile
    ThisHostFile = MyFile

    ' Exclui a antiga página HTML
    On Error Resume Next
    Kill (ThisFile)
    On Error GoTo 0

    ' Constrói o título
    ThisTitle = "<Title>LTCC Membership Directory</Title>"
    WST.Cells(3, 2).Value = ThisTitle

    ' Abre o arquivo para a saída
    Open ThisFile For Output As #1

    ' Escreva a parte superior da HTML
    For j = 2 To FinalT
        Print #1, WST.Cells(j, 2).Value
    Next j

    ' Para cada linha em Membership, escreve linhas de dados para arquivo HTML
    For j = 2 To FinalM
        ' Coloca o nome do Membro entre tags de negrito
        Print #1, "<b>" & WSM.Cells(j, 1).Value & "</b><br>"
        ' Endereço do membro
        Print #1, WSM.Cells(j, 2).Value & "<br>"
        ' Cidade, Estado, CEP
        Addr = WSM.Cells(j, 3) & " " & WSM.Cells(j, 4) & " " & WSM.Cells(j, 5)
        Print #1, Addr & "<br>"
        ' O número de telefone com 2 quebras de linha depois dele
        Print #1, WSM.Cells(j, 6).Value & "<br><br>"
    Next j

    ' Fecha o antigo arquivo
```

```
    ' Escreve a data atualizada, mas certifica-se de que há 20 linhas primeiro
    Print #1, "<br>"
    Print #1, "Essa página foi atualizada em " & Format(Date, "mmmm dd, yyyy") &" " & Format(Time, "h:mm AM/PM")

    ' Escreve o código HTML da planilha Parte inferior
    For j = 2 To FinalB
        Print #1, WSB.Cells(j, 2).Value
    Next j
    Close #1

    Application.StatusBar = False
    Application.CutCopyMode = False
    MsgBox "As páginas Web foram atualizadas"

End Sub
```

A Figura 16.13 mostra a página Web concluída. Ela tem uma aparência muito melhor do que a página genérica criada pela opção Salvar Como Página Web do Excel. Você pode manter a aparência e comportamento do resto do site Web.

Figura 16.13 Um sistema de gerenciamento de conteúdo simples no Excel foi utilizado para gerar essa página Web. A aparência e o comportamento combinam com o resto do site Web. O Excel alcançou esse resultado sem nenhuma codificação de banco de dados de alto custo.

Esse sistema tem muitas vantagens. A pessoa que mantém o diretório de membros se sente à vontade trabalhando no Excel. Ela já tem mantido os dados no Excel regularmente. Agora, depois de atualizar alguns registros, ela pressiona um botão para produzir uma nova versão da página Web.

Evidentemente, o Web designer desconhece o Excel. Mas se ele quiser modificar o design da página Web, bastará abrir seu novo arquivo `sample.html` no Bloco de Notas e copiar o novo código para as planilhas Parte Superior e Parte Inferior.

A página Web resultante tem um tamanho do arquivo pequeno — aproximadamente um sexto do tamanho da página equivalente criada pela funcionalidade do Excel Salvar Como Página Web.

Na vida real, o sistema de gerenciamento de conteúdo desse exemplo foi estendido para permitir a fácil manutenção do calendário, dos membros da diretoria da organização e assim por diante. A pasta de trabalho resultante possibilitou a manutenção de 41 páginas Web com o clique de um botão.

Bônus: FTP no Excel

Depois de conseguir atualizar páginas Web a partir do Excel, você ainda tem o trabalho de utilizar um programa FTP para carregar as páginas da sua unidade de disco para a Internet. Novamente, existem muitas pessoas proficientes em Excel, mas não são tantas as que se sentem à vontade para utilizar um cliente FTP.

Ken Anderson escreveu um excelente utilitário freeware de FTP de linha de comando. Faça o download do WCL_FTP em www.pacific.net/~ken/software/. Salve o WCL_FTP.exe no diretório-raiz da unidade de disco e, depois, utilize este código automaticamente para carregar arquivos HTML recém-criados para o servidor Web:

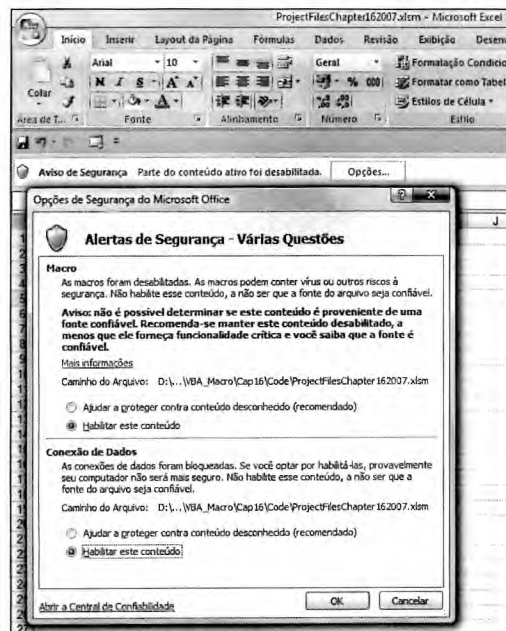
```
Sub DoFTP(fname, pathfname)
'Para esse trabalho, copia wcl_ftp.exe para o diretório-raiz de C:\
'Faz o download a partir de http://www.pacific.net/~ken/software/

' Cria uma string no FTP. A sintaxe é
'WCL_FTP.exe "Legenda" nome do host nome do usuário senha host-diretório
'host-nome do arquivo local-nome do arquivo get-ou-put 0Ascii1Binanry 0NoLog _
'0Background 1CloseWhenDone 1PassiveMode 1ErrorsText

If Not Worksheets("Menu").Range("I1").Value = True Then Exit Sub
```

```
s = """c:\wcl_ftp.exe "" "& """Upload File to website"" "& "ftp.MySite.com FTPUser FTPPassword www "& fname & "
"& """" & pathfname & """ "& "put "& "0 0 0 1 1 1"

Shell s, vbMinimizedNoFocus
End Sub
```

Tornando conteúdo Web confiável

NOVO No Excel 2007, as consultas Web são agrupadas na categoria conteúdo não-confiável. Quando abrimos um arquivo que contém uma consulta Web, ele é automaticamente desativado. Uma mensagem aparece na barra de informações indicando que parte do conteúdo ativo foi desativada.

A pessoa que utiliza a pasta de trabalho poderia clicar no botão Opções para exibir as Múltiplas Questões na pasta de trabalho. O usuário deve ativar tanto todas macros como a conexão de dados, conforme mostrado na Figura 16.14. Ativar a conexão de dados permite que a consulta Web recupere os dados.

Figura 16.14
Quando o Aviso de Segurança for exibido, clique em Opções para ativar as macros e a conexão de dados.

É irritante lidar com o Aviso de Segurança. Se designar como confiável a pasta que contém o seu arquivo, você não será advertido sobre conexões externas de dados toda vez que abrir o arquivo. Para configurar uma localização confiável, siga estes passos:

1. Se o Aviso de Segurança for exibido, clique no link Abrir a Central de Confiabilidade no canto esquerdo inferior do caixa de diálogo e pule o passo 2.

2. Clique no botão Ícone do Office e escolha Opções do Excel. Na barra de navegação esquerda da caixa de diálogo Opções do Excel, escolha Central de Confiabilidade. Clique no botão Configurações da Central de Confiabilidade.

3. Na barra de navegação esquerda da Central de Confiabilidade, escolha Locais Confiáveis. O Excel mostra uma lista de pastas que já são confiáveis (veja Figura 16.15).

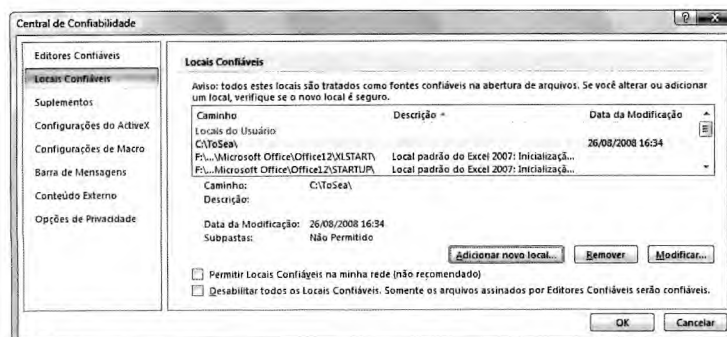
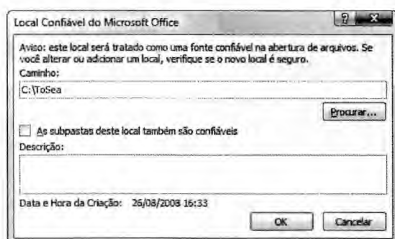
Figura 16.15
Gerencie os locais confiáveis com essa caixa de diálogo.

4. Se o local confiável estiver em uma rede, você deve escolher a caixa de seleção Permitir Locais Confiáveis em Minha Rede. A caixa de diálogo diz que a Microsoft não recomenda isso, mas, se você tiver controle da rede, não há nenhum problema em utilizar um local de rede.
5. Clique no botão Adicionar Novo Local. O Excel exibe a caixa de diálogo Local Confiável do Microsoft Office, como mostrado na Figura 16.16.

Figura 16.16
Adicione um local confiável para impedir que o Excel desative seu conteúdo.

6. Clique no botão Procurar. Navegue para a pasta e clique em OK.
7. Se quiser que todas as subpastas da pasta atual sejam confiáveis, clique na caixa de seleção Subpastas Deste Local Também São Confiáveis.
8. Se quiser documentar a razão pela qual você está tornando essa pasta confiável, digite isso na caixa de texto Descrição.
9. Clique em OK para adicionar a pasta à lista de locais confiáveis. Clique em OK duas ou mais vezes para voltar para o Excel.

   O conteúdo Web e as macros de todos os arquivos armazenados na pasta confiável serão automaticamente ativados.

Próximos passos

O Capítulo 17, "XML no Excel 2007", examina como passar dados entre aplicativos com XML. Alguns sites Web, como o da Amazon, estão oferecendo dados em XML hoje em dia, tornando isso uma ferramenta mais poderosa que as consultas Web.

XML no Excel 2007

17

Um dos recursos introduzidos no Office 2003 era a capacidade de lidar melhor com os dados XML. Se você estiver trabalhando com uma empresa que oferece dados e esquemas XML, as oportunidades são infinitas. Mesmo que empresa não ofereça dados XML, você ainda pode experimentar esses recursos usando dados XML disponíveis publicamente. Mais adiante neste capítulo, examinamos como usar XML para recuperar dados da Amazon.com.

Note que o suporte à XML apareceu apenas em certas edições do Excel 2003, mas era preciso ter a versão completa do Excel 2003 ou o Enterprise Edition do Excel 2003 para usar os recursos de XML. No Excel 2007, a Microsoft disponibilizou a funcionalidade XML em todas as edições.

NESTE CAPÍTULO

O que é XML?

Se você já viu o código-fonte de uma página Web, conhece os tags HTML. Próximo à parte superior da fonte HTML de uma página Web, você verá um tag que define o título de página Web que vai aparecer na barra azul na parte superior da janela do navegador. O tag pode ser semelhante a este:

```
<TITLE>Dicas sobre o Excel do MrExcel</TITLE>
```

Uma página Web pode conter tags para identificar títulos, parágrafos, tabelas, linhas dentro de tabelas e assim por diante. Essa é a linguagem HTML.

XML é como um HTML vitaminado. Com XML, é possível definir absolutamente qualquer campo em XML. O arquivo XML a seguir contém as informações sobre os pedidos de vendas recebidos hoje. Não há nenhuma mágica para criar um arquivo XML; na verdade, digitei isso no Bloco de Notas:

```
<PedidosHoje>
    <PedidoVendas>
        <Cliente>BCA Co</Cliente>
        <Endereço>123 North</Endereço>
        <Cidade>Stow</Cidade>
        <Estado>OH</Estado>
        <CEP>44224</CEP>
        <CodigoItem>23456</CodigoItem>
        <Quantidade>500</Quantidade>
        <PreçoUnidade>21.75</PreçoUnidade>
    </PedidoVendas>
    <PedidoVendas>
        <Cliente>DEF Co</Cliente>
        <Endereço>234 Carapace Lane</Endereço>
        <Cidade>South Bend</Cidade>
        <Estado>IN</Estado>
        <CEP>44685</CEP>
        <CodigoItem>34567</CodigoItem>
        <Quantidade>20</Quantidade>
        <PreçoUnidade>50.00</PreçoUnidade>
    </PedidoVendas>
</PedidosHoje>
```

Regras XML simples

Se estiver familiarizado com HTML, você precisa estar ciente de algumas regras simples que diferenciam XML de HTML:

- Todo elemento de dados tem de iniciar e acabar com um tag idêntico. Os nomes de tag diferenciam letras maiúsculas de minúsculas. `<TagName>Dados</Tagname>` não é válido. `<TagName>Dados</TagName>` é válido.
- O arquivo XML deve iniciar e acabar com um tag-raiz. Há somente um tag-raiz no arquivo. No exemplo anterior, o tag-raiz é `<PedidosHoje>`.
- É válido ter um tag vazio. Coloque uma barra no fim do tag. Se não houver CEP para um pedido internacional, por exemplo, utilize `<CEP/>` a fim de indicar que não há dados para esse campo desse registro.
- Se aninhar tags, o tag interno deve estar fechado antes de você fechar o tag externo. Isso difere de HTML. Em HTML, é válido codificar `<b>XML é bem <i>legal</b></I>`. Isso não é válido em XML. `<Item><a>dados</a></Item>` funcionará, mas não `<Item><a>dados</Item></a>`.

Para uma discussão mais completa sobre XML, veja *XML by Example* (978-0-7897-2504-2), de Benoit Marchal.

Formato de arquivo universal

Durante muitos anos, o CSV (*comma-separated values*) foi considerado o formato de arquivo universal. Quase todos os aplicativos poderiam produzir dados em valores separados por vírgulas e quase todas as planilhas podiam ler dados CSV. O XML promete tornar-se o formato de arquivo universal do futuro, e é muito mais poderoso.

Você pode lidar com dados CSV todos os dias. Como desenvolvedores, precisamos conversar com o outro desenvolvedor que está gerando os dados CSV e entender que a quinta coluna contém um CEP e a sexta um CÓDIGO do produto.

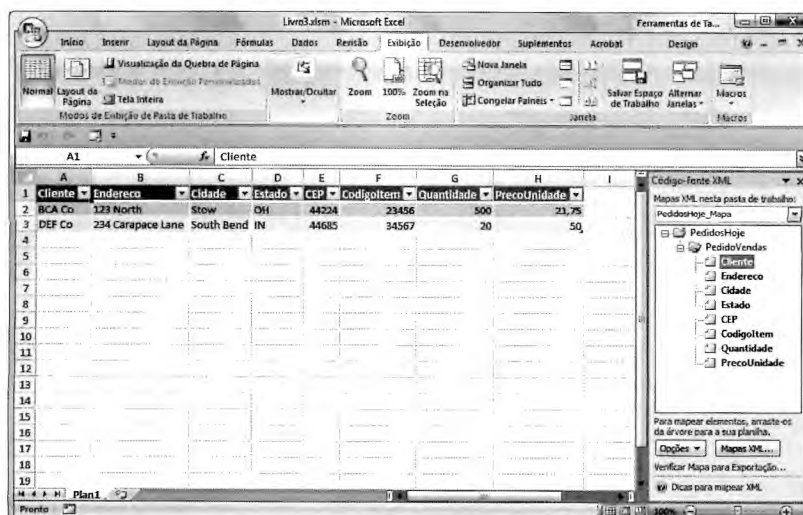
Se o sistema de origem omitir um campo, provavelmente acabaremos com resultados errados. Durante anos, o CSV foi o formato universal para levar dados de outro sistema para uma planilha. Com o CSV, ambos os desenvolvedores devem entender que a quinta coluna contém um CEP e a sexta coluna contém um Codigo do número do produto (veja Figura 17.1).

Figura 17.1
Ao abrir um arquivo CSV, vê-se essa confusa visualização dos dados.

	A	B	C	D	E	F	G	H	I	J
1	BCA Co	123 Noth	Stow	OH	44224	23456	500	21,75		
2	DEF Co	234 Carap	South Ben	IN	44285	34567	20	50		
3										

XML como o novo formato de arquivo universal

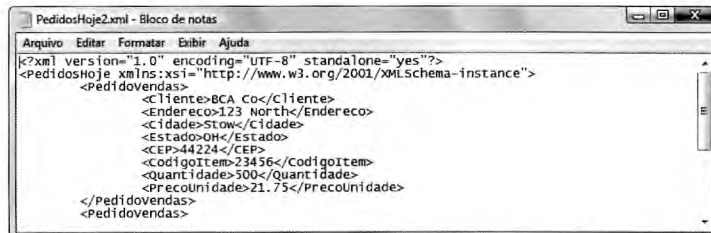
O formato XML é muito mais poderoso. Se você utilizar o Excel para abrir o arquivo XML simples, como o mostrado na Figura 17.1, o Excel pode oferecer títulos e até inferir algo chamado *esquema* dos dados. Um esquema é um arquivo separado que descreve as colunas e relacionamentos inerentes aos dados. Se, por acaso o sistema de fonte não tiver o CEP de um registro, não precisamos nos preocupar se o Código do produto mudar de lugar na coluna F. Compare a Figura 17.2 com o exemplo de CSV mostrado na Figura 17.1. O Excel pode apresentar uma visualização muito mais descritiva dos dados.

Figura 17.2
Os dados podem ser escritos no formato XML por qualquer sistema e transferidos de modo inteligente para o Excel. Diferentemente do formato de dados CSV, o XML é algo que o Excel pode entender.

Se inserir novos registros na lista XML do Excel, você pode salvar o arquivo de volta apenas como dados XML. Utilize o ícone do Office, Salvar Como, Outros Formatos. Na lista suspensa Salvar Como Tipo, escolha Dados XML (*.xml). O Excel pode ler nativamente e gravar para arquivos XML, permitindo que você compartilhe dados entre aplicativos.

Depois de salvar o documento como uma tabela XML, o Excel formata o XML com uma estrutura apropriada e caracteres de tabulação para subelementos. Isso permite que os dados sejam visualizados em um editor de textos simples como o Bloco de Notas (veja Figura 17.3).

Figura 17.3
Depois de adicionar registros aos dados XML no Excel e salvar, o XML resultante é formatado de modo bem organizado.

PedidosHoje2.xml - Bloco de notas
Arquivo Editar Formatar Exibir Ajuda

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PedidosHoje xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
	<PedidoVendas>
		<Cliente>BCA Co</Cliente>
		<Endereco>123 North</Endereco>
		<Cidade>Stow</Cidade>
		<Estado>OH</Estado>
		<CEP>44224</CEP>
		<CodigoItem>23456</CodigoItem>
		<Quantidade>500</Quantidade>
		<PrecoUnidade>21.75</PrecoUnidade>
	</PedidoVendas>
	<PedidoVendas>
```

A sopa de letrinhas do XML

Observe que, no exemplo anterior, tínhamos apenas um arquivo XML e o Excel conseguiu ler os dados com perfeição, permitindo editá-los e gravá-los novamente para utilização por outro aplicativo.

Dois tipos de arquivo adicionais — os esquemas e transformações — aprimoram os arquivos XML.

Embora o arquivo XML contenha os dados e nomes de campos, um arquivo de esquema XML define relacionamentos de dados e requisitos de validação de dados. Por exemplo, um campo CEP poderia requerer cinco dígitos numéricos. Os esquemas XML em geral são armazenados em arquivos XSD.

Os arquivos XSL são chamados *transformações* ou *soluções*. Uma transformação descreve como os campos no arquivo XML devem ser mapeados para o documento. Se seus dados contiverem 20 elementos, é possível definir no arquivo de transformação que você quer ver apenas determinados elementos nessa planilha. Você pode ter muitos arquivos XSL para um determinado esquema a fim de permitir muitas visualizações dos mesmos dados.

Se você estiver lendo dados XML de outra pessoa, provavelmente arquivos XSD e XSL foram oferecidos. Se você estiver lendo seus próprios dados XML, o Excel realmente infere um esquema de dados para você. Depois de abrir o arquivo XML com PedidosHoje introduzido no início deste capítulo, você pode ir para o painel imediato do Editor do VB e recuperar o arquivo XSD. Digite o seguinte:

```
Debug.Print ActiveWorkbook.XmlMaps(1).Schemas(1).xml
```

Copie o resultado em uma janela do Bloco de notas e salve como **`pedidoshoje2.xsd`**. A Figura 17.4 mostra o arquivo XSD inferido, embora eu tenha adicionado quebras de linha e espaçamento para aprimorar a legibilidade.

O passo final para usar integralmente XML no Office 2007 é criar XSL para transformar arquivos. Não há uma maneira fácil de criar isso com as ferramentas no Excel. Para uma discussão do processo excessivamente complicado de criar um arquivo XSL manualmente, veja www.mrexcel.com/tip064.shtml.

Figura 17.4
O Excel é capaz de inferir um esquema padrão para qualquer arquivo XML que ele encontrar. Ter um arquivo de esquema XSD é um requisito para utilizar qualquer recurso XML de mais alto nível, como redirecionamento (repurpose) de dados.

pedidoshoje2.xsd - Bloco de notas
Arquivo Editar Formatar Exibir Ajuda

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"><xsd:element nillable="true"
name="PedidosHoje"><xsd:complexType><xsd:sequence minOccurs="0"><xsd:element minOccurs="0"
maxOccurs="unbounded" nillable="true" name="PedidoVendas"
form="unqualified"><xsd:complexType><xsd:sequence minOccurs="0"><xsd:element minOccurs="0"
nillable="true" type="xsd:string" name="Cliente"
form="unqualified"></xsd:element><xsd:element minOccurs="0" nillable="true" type="xsd:string"
name="Endereco" form="unqualified"></xsd:element><xsd:element minOccurs="0" nillable="true"
type="xsd:string" name="Cidade" form="unqualified"></xsd:element><xsd:element minOccurs="0"
nillable="true" type="xsd:string" name="Estado" form="unqualified"></xsd:element><xsd:element
minOccurs="0" nillable="true" type="xsd:integer" name="CEP"
form="unqualified"></xsd:element><xsd:element minOccurs="0" nillable="true"
type="xsd:integer" name="CodigoItem" form="unqualified"></xsd:element><xsd:element
minOccurs="0" nillable="true" type="xsd:integer" name="Qu
antidade" form="unqualified"></xsd:element><xsd:element minOccurs="0" nillable="true"
type="xsd:double" name="PrecoUnidade"
form="unqualified"></xsd:element></xsd:sequence></xsd:complexType></xsd:element></xsd:sequenc
e></xsd:complexType></xsd:element></xsd:schema>
```

NOVO O uso de XML como tipo de arquivo pela Microsoft

A Microsoft tem aprimorado rapidamente seu suporte a XML:

- No Excel 2002, inicialmente era impossível ler um arquivo XML.
- No Excel 2003, era possível escrever arquivos XML. Essa também era a primeira versão em que se podia optar por salvar arquivos do Excel ou do Word como XML em vez dos usuais tipos de arquivo binário XLS ou DOC. Na edição anterior deste livro, nos admiramos com a capacidade de o Excel fazer uma viagem de ida e volta (*round-trip*) entre diferentes arquivos, convertendo um arquivo para XML e, depois, convertendo de novo para o Excel sem perder nenhuma fórmula ou formatação (veja Figura 17.5). No Excel 2003, o tipo de arquivo XML não suportaria gráficos ou VBA.

Figura 17.5
No Excel 2003, a maioria dos elementos de planilha, exceto gráficos e VBA, eram suportados pelo tipo de arquivo XML.

TOP 20 CUSTOMERS IN THE WEST REGION

Customer	Revenue	Gross Profit	GP%
RST INC.	5,017K	2,776K	55.3%
RST S.A.	4,311K	2,462K	57.1%
WXY, CO	3,500K	1,963K	56.1%
LMN INC.	3,500K	1,953K	55.8%
VWX GMBH	3,486K	1,961K	56.3%
OPQ, INC.	3,248K	1,818K	56.0%
HIJ GMBH	2,491K	1,373K	55.1%
MNO CORP	2,325K	1,294K	55.7%
CDE INC.	2,287K	1,245K	54.4%
DEF, LLC	1,714K	962K	56.1%
BCD LTD.	904K	493K	54.5%
LMN LTD.	735K	424K	57.6%
FGH LTD.	730K	401K	54.9%
VWX PTY LTD	723K	413K	57.1%
EFG S.A.	568K	291K	51.3%
TUV GMBH	557K	303K	54.4%
LMN PTY LTD	531K	306K	57.6%

- No Excel 2007, o tipo de arquivo padrão para os arquivos do Excel é um arquivo XML compactado com um utilitário zip. A próxima seção descreve como o Excel 2007 armazena pastas de trabalho com XML.

Como o Excel 2007 armazena pastas de trabalho com XML

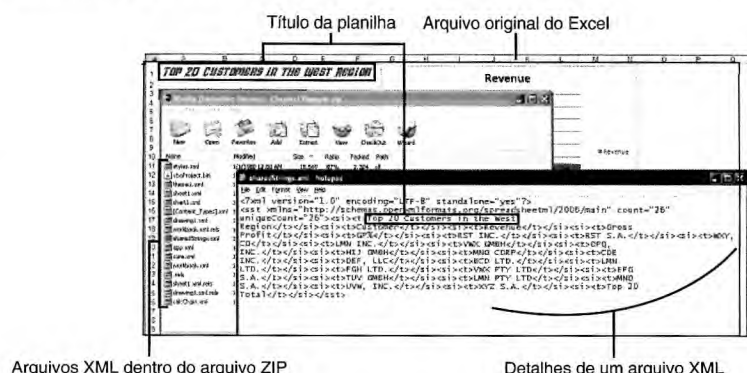
O Excel 2007 introduz novos formatos de arquivo, como XLSX, XLSM e XLSB. O último formato de arquivo é ainda um formato binário proprietário. Mas os formatos de arquivo XLSX e XLSM são cem por cento formatos XML. O Excel salva a pasta de trabalho, incluindo gráficos, elementos gráficos SmartArt, fórmulas, formatação e números em vários arquivos XML. Todos os formatos XML são, então, compactados com zip em um único arquivo e a extensão é alterada de .zip para. xlsk.

Além disso, o formato XLSM inclui suporte para macros, módulos e userforms.

Você pode explorar facilmente o funcionamento interno de um arquivo do Excel 2007 seguindo estes passos.

1. Crie um arquivo Excel. Adicione fórmulas, formatação, um gráfico e algum código de macro VBA.
2. Salve o arquivo como um arquivo XLSM.
3. Feche o arquivo.
4. Utilizando o Windows Explorer, navegue para a pasta que contém o arquivo.
5. Renomeie o arquivo para ter uma extensão .zip. Você pode ser advertido de que isso pode tornar o arquivo inutilizável. Está tudo certo, não se preocupe.
6. Abra o arquivo com WinZip ou seu utilitário favorito de descompactação de arquivo zip. Você verá que o arquivo é composto de muitas partes de XML. O gráfico e o VBA terão sua própria parte de XML. Você terá uma parte de XML para cada planilha. A árvore de cálculo está agora completamente exposta e explorável em CalcChain.xml.

A Figura 17.6 mostra a pasta de trabalho original, os arquivos no arquivo zip e uma das partes do XML. O Excel armazena strings compartilhadas aqui a fim de diminuir o tamanho do arquivo.

Figura 17.6
No Excel 2007, a Microsoft armazena a maioria das partes da planilha em arquivos XML de texto simples e os compacta na forma de um único arquivo zip.

Estudo de caso

Utilizando dados XML da Amazon.com

A promessa é de que todos os tipos de dados XML logo estarão disponíveis. Um exemplo de alto perfil disponível hoje é o da Amazon.com. A empresa oferece uma ampla variedade de dados consultáveis que são retornados em formato XML.

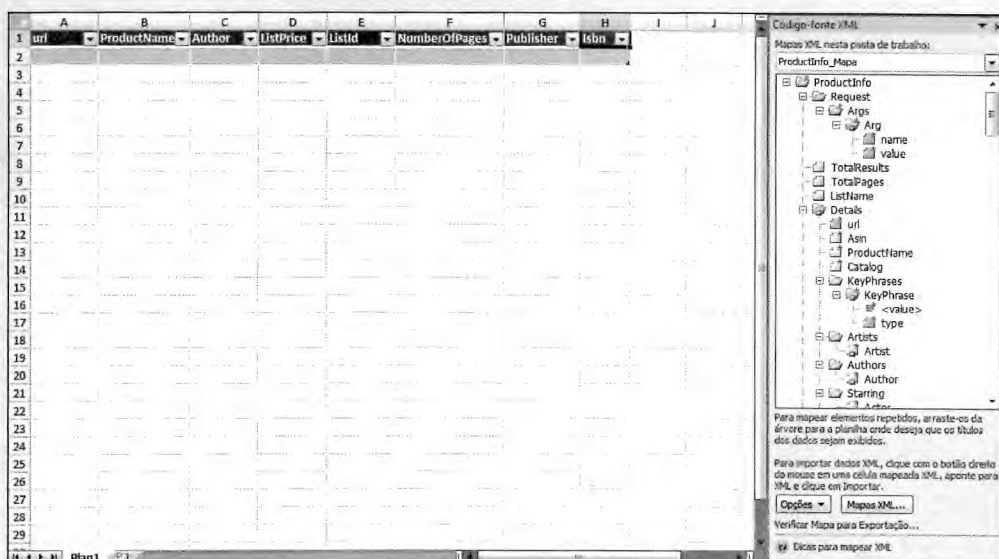
É possível utilizar o VBA para consultar dados da Amazon e obter os resultados no Excel.

A Amazon.com oferece mais de 30 streams XML diferentes, que você pode consultar e utilizar para completar a documentação de informações disponíveis em: `http://amazon.com/webservices`.

Iniciando com uma pasta de trabalho em branco do Excel, primeiro anexe o esquema à pasta de trabalho:

1. Na guia Desenvolvedor da faixa, selecione Código-Fonte no grupo XML.
2. No painel Código-Fonte XML, clique no botão Mapas XML.
3. Na caixa de diálogo Mapas XML, clique no botão Adicionar.
4. Na caixa Nome do Arquivo, digite o URL para o pesado arquivo XSD da Amazon: `http://xml.amazon.com/schemas3/dev-heavy.xsd. Select Open.`
5. Na caixa de diálogo Múltiplas Raízes, selecione ProductInfo como a nota-raiz de interesse.
6. Clique em OK para fechar a caixa de diálogo Mapas XML.

Todos os campos disponíveis no esquema ProductInfo da Amazon.com aparecerão no painel Código-Fonte XML. Arraste todos os campos de interesse para sua planilha, como mostrado na Figura 17.7.

Figura 17.7
Depois que o esquema XSD for anexado à pasta de trabalho, você pode arrastar campos de interesse do painel Código-Fonte XML à planilha.

Agora que a pasta de trabalho está configurada, é possível retornar ao VBA para inserir o código necessário para recuperar dados da Amazon.com.

O primeiro passo é ir para Ferramentas, Referências, e adicionar uma referência a Microsoft WinHTTP Services.

O Amazon Development Kit fornece muitas strings URL de exemplo para retornar pesquisas por autor, editor, ISBN e assim por diante.

Depois de definir a string adequada de URL, ela é enviada como um WinHttpRequest e os resultados são mapeados para a planilha com o método `.ImportXML`.

Esse código de exemplo consulta na Amazon todos os livros de Bill Jelen publicados em 2006:

```
Sub QuerybyAuthor()
    ' Consulta todos os livros de Bill Jelen publicados em 2006
    Dim ws As Worksheet
    Dim whr As New WinHttpRequest
    Dim lobj As ListObject
    Dim nCount As Integer
    Dim objSelected As Object

    Dim sURI As String
```

```
    Set objSelected = Selection
    Set ws = Worksheets("Sheet1")
    Set lobj = ws.ListObjects(1)

    Application.Cursor = xlWait

    sURI = "http://xml.amazon.com/onca/xml2?t=webservices-20"& "&dev-t=D3VCC047XZEQFA"& _
      "&PowerSearch=author:Bill Jelen and pubdate:2006"& "&mode=books&type=heavy&page=1&f=xml"

    whr.Open "GET", sURI
    whr.Send

    ActiveWorkbook.XmlMaps(1).ImportXml whr.ResponseText

    Application.Cursor = xlDefault

End Sub
```

Depois que a consulta tiver sido feita, os resultados serão gravados no objeto List em Sheet1. Observe que, como solicitei um campo Autor, e Autor é um elemento que se repete ao longo do nó ProductInfo, todos os títulos com dois autores aparecem duas vezes, um com cada autor (veja Figura 17.8).

	B	C	D	E
1	ProductName	Author	ListPrice	ListId
2	Special Edition Using Microsoft(R) Office Excel 2007	Bill Jelen	$39.99	
3	Special Edition Using Microsoft(R) Office Excel 2007		$39.99	R1PT8TV1XR9EDD
4	Special Edition Using Microsoft(R) Office Excel 2007		$39.99	R3TRNUNDTL80S5
5	Special Edition Using Microsoft(R) Office Excel 2007		$39.99	R2NQ78N8365PHX
6	Pivot Table Data Crunching for Microsoft(R) Office Excel(R) 2007 (Business Solutions)	Bill Jelen	$29.99	
7	Pivot Table Data Crunching for Microsoft(R) Office Excel(R) 2007 (Business Solutions)	Michael Alexander	$29.99	
8	Excel 2007 Miracles Made Easy: Mr. Excel Reveals 25 Amazing Things You Can Do with th	Bill Jelen	$24.95	
9	Excel 2007 Miracles Made Easy: Mr. Excel Reveals 25 Amazing Things You Can Do with the New Excel		$24.95	R1PT8TV1XR9EDD
10	Holy Macro! It's 2,500 Excel VBA Examples: Every Snippet of Excel VBA Code You'll Ever N	Hans Herber	$89.00	
11	Holy Macro! It's 2,500 Excel VBA Examples: Every Snippet of Excel VBA Code You'll Ever N	Bill Jelen	$89.00	
12	Holy Macro! It's 2,500 Excel VBA Examples: Every Snippet of Excel VBA Code You'll Ever N	Tom Urtis	$89.00	
13	Excel for Marketing Managers (Excel for Professionals series)	Ivana Taylor	$24.95	
14	Excel for Marketing Managers (Excel for Professionals series)	Bill Jelen	$24.95	
15	Excel for Marketing Managers (Excel for Professionals series)		$24.95	2UQPX2L1T9ISA

Figura 17.8
O método `.ImportXML` retorna os resultados da pesquisa à planilha. Apenas os campos mapeados que foram arrastados do painel Código-Fonte XML são exibidos.

É interessante observar que o ProductInfo_Map oferece dezenas de campos e os resultados da consulta podem ser facilmente alterados arrastando-se e soltando-se novos dados do painel Código-Fonte XML na pasta de trabalho. Sem mudar uma única linha de código do VBA, a consulta pode facilmente focalizar SalesRank, Lists ou o número de vendedores que oferecem o livro para venda.

Próximos passos

Embora o XML permita transferir dados entre aplicativos não relacionados, você já pode transferir dados programaticamente entre aplicativos no conjunto do Microsoft Office. O Capítulo 18, "Automatizando o Word", demonstra como utilizar o Excel VBA para automatizar e controlar o Microsoft Word.

Automatizando o Word

18

NESTE CAPÍTULO

O Word, o Excel, o PowerPoint, o Outlook e o Access utilizam a mesma linguagem VBA; a única diferença entre eles são seus modelos de objeto (por exemplo, o Excel tem um objeto `Workbooks`, o Word tem `Documents`). Qualquer um desses aplicativos pode acessar o modelo de objeto do outro aplicativo contanto que o segundo aplicativo esteja instalado.

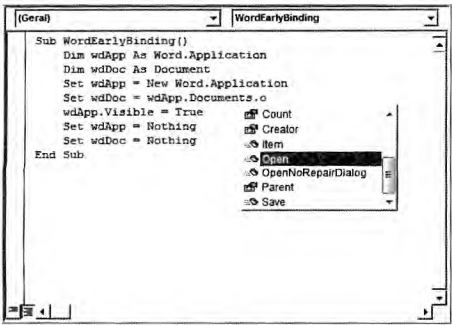
Para acessar a biblioteca de objetos do Word, o Excel deve estabelecer um vínculo com ela. Há duas maneiras de fazer isso: a vinculação inicial ou a vinculação tardia. Com a vinculação inicial, a referência para o objeto do aplicativo é criada quando o programa é compilado; com a vinculação tardia, a referência é criada quando o programa é executado.

Este capítulo é uma introdução para acessar o Word a partir do Excel; não revisaremos todo o modelo de objeto do Word nem o de outros aplicativos. Consulte o Navegador de Objetos do VBA no aplicativo apropriado para conhecer outros modelos de objeto.

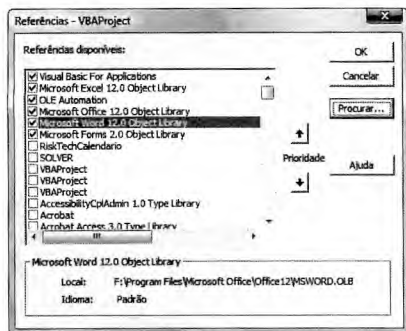
Vinculação inicial

O código escrito com a vinculação inicial executa mais rapidamente do que aquele com a vinculação tardia. Uma referência é feita à biblioteca de objetos do Word, antes de o código ser escrito, para que os objetos, propriedades e métodos do Word estejam disponíveis no Navegador de Objetos. As dicas também aparecem, conforme mostrado na Figura 18.1, como uma lista de membros de um objeto.

A desvantagem da vinculação inicial é que a biblioteca de objetos referenciada deve existir no sistema. Por exemplo, se você escrever uma macro para referenciar a biblioteca de objetos do Word 2007 e um usuário do Word 2003 tentar executar o código, o programa falhará porque não conseguirá localizar a biblioteca de objetos 2007.

Figura 18.1
A vinculação inicial permite acesso fácil à sintaxe de objeto do Word.

A biblioteca de objetos é adicionada pelo Editor do VB:

1. Selecione Ferramentas, Referências.
2. Marque Microsoft Word 12.0 Object Library na lista Referências Disponíveis (veja Figura 18.2).
3. Clique em OK.

Figura 18.2
Selecione a biblioteca de objetos na lista Referências.

> **NOTA** Se a biblioteca de objetos não for localizada, o Word não está instalado. Se outra versão for localizada na lista (como 10.0), outra versão do Word está instalada.

Depois que a referência estiver configurada, as variáveis do Word podem ser declaradas com o tipo correto de variável (Word). Mas se a variável de objeto for declarada `As Object`, isso forçará o programa a utilizar a vinculação tardia:

```
Sub WordEarlyBinding()
Dim wdApp As Word.Application
Dim wdDoc As Document
Set wdApp = New Word.Application
Set wdDoc = wdApp.Documents.Open(ThisWorkbook.Path &"\Capítulo 18 - Automatizando o Word.docx")
wdApp.Visible = True
Set wdApp = Nothing
Set wdDoc = Nothing
End Sub
```

> **DICA** O Excel pesquisa pelas bibliotecas selecionadas para localizar a referência para o tipo de objeto. Se o tipo for localizado em mais de uma biblioteca, a primeira referência será selecionada. Você pode definir qual biblioteca deve ser escolhida alterando a prioridade da referência na listagem.

Esse exemplo cria uma nova instância do Word e abre um documento do Word existente a partir do Excel. As variáveis declaradas, `wdApp` e `wdDoc`, são de tipos de objeto Word. A variável `wdApp` é utilizada para criar uma referência ao aplicativo Word da mesma maneira que o objeto `Application` é utilizado no Excel. `New Word.Application` é utilizado para criar uma nova instância do Word.

> **DICA** Se você estiver abrindo um documento em uma nova instância do Word, este programa não será visível. Se for necessário mostrar o aplicativo, ele deverá ser reexibido (`wdApp.Visible = True`).

Quando concluído, é uma boa idéia configurar as variáveis de objeto como `Nothing` e liberar a memória para ser utilizada pelo aplicativo, como mostrado aqui:

```
Set wdApp = Nothing
Set wdDoc = Nothing
```

Erro de compilação: não foi possível localizar o objeto nem a biblioteca

Se a versão referenciada do Word não existir no sistema, uma mensagem de erro aparecerá, como mostrado na Figura 18.3. Visualize a lista Referências; o objeto ausente é realçado com a palavra AUSENTE (Veja Figura 18.4).

Figura 18.3
Tentar compilar um programa com uma biblioteca de referência ausente gerará uma mensagem de erro.

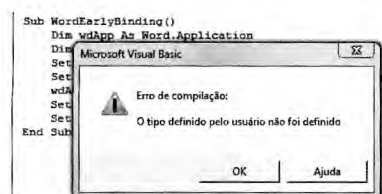
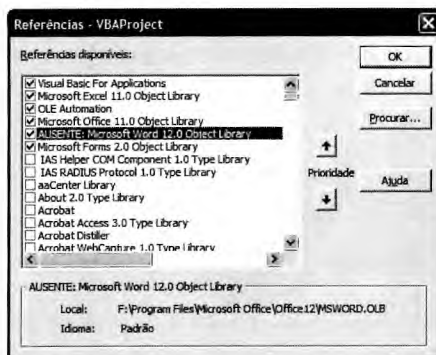



Figura 18.4
O Excel listará a biblioteca ausente para você.

Se uma versão anterior do Word estiver disponível, você pode tentar executar o programa com essa versão referenciada. Muitos objetos são os mesmos entre versões diferentes.

Vinculação tardia

Ao utilizar a vinculação tardia, você está criando um objeto que se refere ao aplicativo Word antes de vincular-se à biblioteca Word. Como você não configura uma referência de antemão, a única restrição na versão Word é que os objetos, propriedades e métodos devem existir. Caso existam diferenças, a versão pode ser verificada e o objeto correto utilizado de acordo.

A desvantagem da vinculação tardia é que o Excel não sabe o que está acontecendo — não entende que você está se referindo ao Word. Isso impede que as dicas sejam exibidas ao referenciar objetos do Word. Além disso, constantes predefinidas não estão disponíveis e, ao compilar, o Excel não pode verificar se as referências ao Word são corretas. Depois de o programa ser executado, os links para o Word começam a ser construídos e todos os erros de codificação são detectados nesse ponto.

O exemplo seguinte cria uma nova instância do Word, abrindo e tornando visível um documento do Word existente:

```
Sub WordLateBinding()
Dim wdApp As Object, wdDoc As Object
Set wdApp = CreateObject("Word.Application")
Set wdDoc = wdApp.Documents.Open(ThisWorkbook.Path & _
"\Capítulo 18 - Automatizando o Word.docx")
wdApp.Visible = True
Set wdApp = Nothing
Set wdDoc = Nothing
End Sub
```

Uma variável de objeto (`wdApp`) é declarada e configurada para referenciar o aplicativo (`CreateObject("Word.Application")`). Outras variáveis requeridas são declaradas (`wdDoc`) e o objeto aplicativo é utilizado para referenciar essas variáveis ao modelo de objetos do Word.

ATENÇÃO

Declarar `wdApp` e `wdDoc` como objetos força o uso da vinculação tardia. O programa não pode criar os links necessários para o modelo de objetos do Word até ele executar a função `CreateObject`.

Criando e referenciando objetos

As seções a seguir descrevem como criar novos objetos bem como referenciar objetos atualmente abertos.

Palavra-chave New

No exemplo de vinculação inicial, a palavra-chave `New` foi utilizada para referenciar o aplicativo Word. Essa palavra-chave pode ser utilizada apenas com vinculação inicial; não funciona com vinculação tardia. `CreateObject` ou `GetObject` também funcionariam, mas `New` era melhor para o exemplo acima. Se uma instância do aplicativo estiver sendo executada e você quiser utilizá-la, em vez disso utilize a função `GetObject`.

ATENÇÃO

Se o código para abrir o Word executar suavemente, mas você não localizar uma instância do Word (e deveria localizar), abra seu Gerenciador de Tarefas e procure o processo WinWord.exe. Se ele existir, na janela Verificação Imediata do Editor do VB do Excel, digite o seguinte (vinculação inicial):

```
Word.Application.Visible = True
```

Se várias instâncias do WinWord.exe forem localizadas, você deve torná-las visíveis e fechar a(s) instância(s) extra(s) WinWord.exe.

Função `CreateObject`

A função `CreateObject` foi utilizada no exemplo de vinculação tardia, mas também pode ser utilizada na vinculação inicial. `CreateObject` tem um parâmetro `class` que consiste do nome e tipo do objeto a ser criado (`Name.Type`). No caso dos exemplos que mostrei (`Word.Application`), Word é o `Name` e Aplicativo é o `Type`.

Ele cria uma nova instância do objeto; nesse caso, o aplicativo Word é criado.

Função `GetObject`

A função `GetObject` pode ser utilizado para referenciar uma instância do Word que já esteja em execução. Ele falhará se não for possível localizar nenhuma instância.

Os dois parâmetros do `GetObject` são opcionais. O primeiro parâmetro especifica o caminho completo e o nome de arquivo a ser aberto e o segundo parâmetro especifica o programa aplicativo.

No exemplo seguinte, deixamos o aplicativo de lado, permitindo que o programa-padrão (que é o Word) abra o documento:

```
Sub UseGetObject()
Dim wdDoc As Object
Set wdDoc = GetObject(ThisWorkbook.Path & "\Capítulo 18 - Automatizando o Word.docx")
wdDoc.Application.Visible = True
Set wdDoc = Nothing
End Sub
```

Esse exemplo abre um documento em uma instância existente do Word e assegura que propriedade `Visible` do aplicativo Word seja configurada como `True`. Observe que, para tornar o documento visível, você tem de referir-se ao objeto aplicativo (`wdDoc.Application.Visible`), porque `wdDoc` está referenciando um documento em vez de um aplicativo.

NOTA

Embora a propriedade `Visible` do aplicativo Word seja configurada como `True`, esse código não torna o aplicativo Word o aplicativo ativo. Na maioria dos casos, o ícone do aplicativo Word permanece na barra de tarefas e o Excel permanece o aplicativo ativo na tela do usuário.

O exemplo a seguir utiliza erros para aprender se o Word já está aberto ou não antes de colar um gráfico no fim de um documento. Se não estiver, ele abre o Word e cria um novo documento:

```
Sub IsWordOpen()
Dim wdApp As Word.Application

ActiveChart.ChartArea.Copy

On Error Resume Next
Set wdApp = GetObject(, "Word.Application")
If wdApp Is Nothing Then
    Set wdApp = GetObject("", "Word.Application")
    With wdApp
        .Documents.Add
        .Visible = True
    End With
End If
On Error GoTo 0

With wdApp.Selection
    .EndKey Unit:=wdStory
    .TypeParagraph
    .PasteSpecial Link:=False, DataType:=wdPasteOLEObject,Placement:=wdInLine, DisplayAsIcon:=False
End With

Set wdApp = Nothing
End Sub
```

Utilizar `On Error Resume Next` força o programa a continuar, mesmo que ele depare com um erro. Nesse caso, um erro ocorre quando se tenta vincular `wdApp` a um objeto que não existe. `wdApp`, então, não terá nenhum valor.

A próxima linha, `If wdApp Is Nothing then`, tira proveito disso e abre uma instância do Word, adicionando um documento vazio e tornando o aplicativo visível. Note o uso de aspas vazias para o primeiro parâmetro em `GetObject('", "Word.Application")` — isso é como utilizar a função `GetObject` para abrir uma nova instância do Word. Utilize `On Error Goto 0` para retornar ao comportamento normal da forma de tratamento do VBA.

> **NOTA** Você pode configurar a biblioteca de referência do Word para acessá-la a partir do Navegador de Objetos, mas não precisa configurar o código com vinculação inicial. Dessa maneira, você pode ter a referência prontamente disponível, mas o código ainda será de vinculação tardia. A desativação da biblioteca de referência é facilmente acessível com alguns poucos cliques.

Utilizando valores constantes

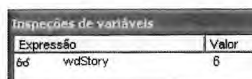
No exemplo anterior, utilizamos constantes que são específicas ao Word, como `wdPasteOLEObject` e `wdInLine`. Quando você programa utilizando a vinculação inicial, o Excel o ajuda mostrando essas constantes na janela de dicas.

Com a vinculação tardia, essas dicas não aparecerão. O que você pode fazer? Você poderia escrever o programa utilizando a vinculação inicial e, em seguida, depois de compilá-lo e testá-lo, mudar para a vinculação tardia. O problema com esse método é que ele não será compilado, porque o Excel não reconhece as constantes do Word.

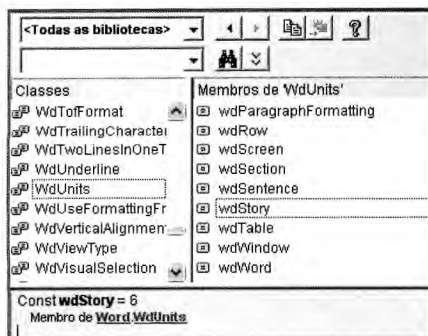
As palavras *wdPasteOLEObject* e *wdInLine* são para sua conveniência como programador. Por trás de cada uma dessas constantes de texto o que o VBA entende é o valor real. A solução para isso é recuperar e utilizar esses valores reais com o programa de vinculação tardia.

Utilizando a janela Inspeção para recuperar o valor real de uma constante

Uma maneira de recuperar o valor é adicionar uma inspeção às constantes. Então, percorra o código e verifique o valor da constante quando ela aparecer na janela Inspeção de Variáveis, como mostrado na Figura 18.5.

Figura 18.5
Utilize a janela Inspeção de Variáveis para obter o valor real por trás de uma constante do Word.

Utilizando o Navegador de Objetos para recuperar o valor real de uma constante

Um segundo método é pesquisar a constante no Navegador de Objetos. Você precisará da biblioteca Word configurada como uma referência. Clique com o botão direito do mouse na constante, selecione Definição e o Navegador de Objetos abrirá a constante, exibindo o valor na janela da parte inferior, conforme mostrado na Figura 18.6.

Figura 18.6
Utilize o Navegador de Objetos para obter o valor real por trás de uma constante do Word.

Substituir as constantes no exemplo de código anterior com seus valores reais seria parecido com isto:

```
With wdApp.Selection
    .EndKey Unit:=6
    .TypeParagraph
    .PasteSpecial Link:=False, DataType:=0,Placement:=0, DisplayAsIcon:=False
End With
```

Mas o que acontecerá daqui a um mês, quando você retornar o código e tentar lembrar-se do que esses números significam? A solução fica a seu critério. Alguns programadores apenas adicionam comentários ao código que referencia a constante Word. Outros, criam suas próprias variáveis para armazenar o valor real e utilizam essas variáveis no lugar da constante, como isto:

```
Const xwdStory As Long = 6
Const xwdPasteOLEObject As Long = 0
Const xwdInLine As Long = 0

With wdApp.Selection
    .EndKey Unit:=xwdStory
    .TypeParagraph
    .PasteSpecial Link:=False, DataType:=xwdPasteOLEObject,Placement:=xwdInLine, DisplayAsIcon:=False
End With
```

Entendendo objetos do Word

O programa de gravação de macros do Word pode ser utilizado para se obter um entendimento preliminar do modelo de objetos do Word. Mas, de modo muito semelhante ao programa de gravação de macros do Excel, os resultados serão enfadonhos. Tenha isso em mente e utilize o gravador para conduzir em direção a objetos, propriedades e métodos no Word.

ATENÇÃO

O programa de gravação de macros é limitado em relação ao que ele permitirá que você grave. O mouse não pode ser utilizado para mover o cursor nem selecionar objetos, mas não há nenhum limite em fazer isso com o teclado.

O próximo exemplo é o que o programa de gravação de macros do Word produz ao adicionar um novo documento em branco.

```
Documents.Add Template:="Normal", NewTemplate:=False, DocumentType:=0
```

Tornar isso mais eficiente (ainda no Word) produz isto:

```
Documents.Add
```

`Template`, `NewTemplate` e `DocumentType` são todas as propriedades opcionais que o gravador inclui, mas não são necessárias a menos que você precise alterar uma propriedade-padrão ou assegurar que uma propriedade é como você requer.

Para utilizar a mesma linha de código no Excel, um link para a biblioteca de objetos do Word é necessário, como vimos anteriormente. Depois que esse link é estabelecido, um entendimento dos objetos do Word é tudo de que você precisa. A seguir uma revisão de alguns dos objetos do Word — o suficiente para fazê-lo decolar. Para uma listagem mais detalhada, refira-se ao modelo de objetos no Editor do VB do Word.

Objeto `Document`

O objeto `Document` do Word é o equivalente ao objeto `Workbook` do Excel. O objeto `Document` consiste de caracteres, palavras, frases, parágrafos, seções e cabeçalhos/rodapés. É por meio do objeto `Document` que são utilizados métodos e propriedades que afetam o documento inteiro, como imprimir, fechar, pesquisar e revisar.

Crie um novo documento em branco

Para criar um documento em branco em uma instância existente do Word, utilize o método `Add` (já aprendemos a criar um novo documento quando o Word é fechado — refira-se a `GetObject` e `CreateObject`):

```
Sub NewDocument()
Dim wdApp As Word.Application

Set wdApp = GetObject(, "Word.Application")

wdApp.Documents.Add

Set wdApp = Nothing
End Sub
```

Esse exemplo abre um novo documento em branco que utiliza o modelo-padrão. Para criar um novo documento com um modelo específico, utilize isto:

```
wdApp.Documents.Add Template:="Contemporary Memo.dot"
```

Isso cria um novo documento que se vale do modelo Contemporary Memo. `Template` pode ser apenas o nome de um modelo do local do modelo-padrão ou o caminho e nome do arquivo.

Abra um documento existente

Para abrir um documento existente, utilize o método `Open`. Vários parâmetros estão disponíveis, inclusive `Read Only` e `AddtoRecentFiles`. O exemplo seguinte abre um documento existente como `Read Only`, mas impede que o arquivo seja adicionado à Lista Recente de Arquivos no menu Arquivo:

```
wdApp.Documents.OpenFilename:="C:\Excel VBA 2007, de Jelen & Syrstad\Cap. 19 - Matrizes.docx",ReadOnly:=True, _
AddtoRecentFiles:=False
```

Salve as alterações em um documento

Depois de fazer alterações em um documento, muito provavelmente você vai querer salvá-lo. Para salvar um documento com o nome existente, utilize isto:

```
wdApp.Documents.Save
```

Se o comando Save for utilizado com um novo documento sem nome, a caixa de diálogo Salvar Como aparecerá. Para salvar um documento com um novo nome, você pode utilizar o método SaveAs :

```
wdApp.ActiveDocument.SaveAs "C:\Excel VBA 2007, de Jelen & Syrstad\MemoTest.docx"
```

`SaveAs` requer o uso de membros do objeto `Document`, como `ActiveDocument`.

Feche um documento aberto

Utilize o método `Close` para fechar um documento especificado ou todos os documentos abertos. Por padrão, um caixa de diálogo Salvar aparece para qualquer documento com alterações que não tiverem sido salvas. O argumento `SaveChanges` pode ser utilizado para mudar isso. Para fechar todos os documentos abertos sem salvar as alterações, utilize esse código:

```
wdApp.Documents.Close SaveChanges:=wdDoNotSaveChanges
```

Para fechar um documento específico, você pode fechar o documento ativo ou especificar um nome de documento:

```
wdApp.ActiveDocument.Close
```

ou

```
wdApp.Documents("Capítulo 19 - Matrizes.docx").Close
```

Imprima um documento

Utilize o método `PrintOut` para imprimir parte de um documento ou um documento inteiro. Para imprimir um documento com todas as configurações de impressão-padrão, utilize isto:

```
wdApp.ActiveDocument.PrintOut
```

Por padrão, o intervalo de impressão é o documento inteiro, mas isso pode ser alterado configurando-se os argumentos `Range` e `Pages` do método `PrintOut`:

```
wdApp.ActiveDocument.PrintOut Range:=wdPrintRangeOfPages, Pages:="2"
```

Objeto `Selection`

O objeto `Selection` representa o que está selecionado no documento — isto é, uma palavra, frase ou o ponto de inserção. Esse objeto tem uma propriedade `Type` que retorna o tipo do que estiver selecionado (`wdSelectionIP`, `wdSelectionColumn`, `wdSelectionShape` e assim por diante).

`HomeKey/EndKey`

Os métodos `HomeKey` e `EndKey` são utilizados para alterar a seleção; eles correspondem à utilização das teclas Home e End, respectivamente, no teclado. Você tem dois parâmetros: `Unit` e `Extend`. `Unit` é o intervalo do movimento a ser feito, para o início (`Home`) ou final (`End`) de uma linha (`wdLine`), documento (`wdStory`), coluna (`wdColumn`) ou linha (`wdRow`). `Extend` é o tipo de movimento: `wdMove` move a seleção, `wdExtend` estende a seleção do ponto de inserção original ao novo ponto de inserção.

Para mover o cursor para o começo do documento, utilize este código:

```
wdApp.Selection.HomeKey Unit:=wdStory, Extend:=wdMove
```

Para selecionar o documento do ponto de inserção até o fim do documento, utilize este código:

```
wdApp.Selection.EndKey Unit:=wdStory, Extend:=wdExtend
```

TypeText

O método `TypeText` é utilizado para inserir texto em um documento do Word. As configurações de usuário, como a configuração `Overtype`, podem afetar o que acontece quando o texto é inserido no documento:

```
Sub InsertText()
Dim wdApp As Word.Application
Dim wdDoc As Document
Dim wdSln As Selection

Set wdApp = GetObject(, "Word.Application")
Set wdDoc = wdApp.ActiveDocument
Set wdSln = wdApp.Selection

wdDoc.Application.Options.Overtype = False
With wdSln
    If .Type = wdSelectionIP Then
        .TypeText ("Inserindo um ponto de inserção. ")
    ElseIf .Type = wdSelectionNormal Then
            If wdApp.Options.ReplaceSelection Then
                .Collapse Direction:=wdCollapseStart
            End If
           .TypeText ("Inserindo antes um bloco de texto. ")
    End If
End With
Set wdApp = Nothing
Set wdDoc = Nothing
End Sub
```

Objeto Range

O objeto `Range` utiliza a seguinte sintaxe:

```
Range(StartPosition, EndPosition)
```

O objeto `Range` representa uma área contígua, ou áreas, no documento. Ele tem uma posição de caractere inicial e uma posição de caractere final. O objeto pode ser o ponto de inserção, um intervalo de texto ou o documento inteiro, incluindo caracteres não-imprimíveis (como espaços ou marcas de parágrafo).

O objeto `Range` é semelhante ao `Selection`, mas melhor: exige menos código para realizar as mesmas tarefas, tem mais capacidades e economiza tempo e memória, pois não requer que o Word mova o cursor ou realce objetos no documento para manipulá-los.

Defina um intervalo

Para definir um intervalo, insira uma posição inicial e final, como mostrado neste segmento de código:

```
Sub RangeText()
Dim wdApp As Word.Application
Dim wdDoc As Document
Dim wdRng As Word.Range

Set wdApp = GetObject(, "Word.Application")
Set wdDoc = wdApp.ActiveDocument

Set wdRng = wdDoc.Range(0, 22)
wdRng.Select

Set wdApp = Nothing
Set wdDoc = Nothing
Set wdRng = Nothing
End Sub
```

A Figura 18.7 mostra os resultados da execução desse código. Os primeiros 22 caracteres, incluindo caracteres não-imprimíveis como retornos de parágrafo, são selecionados.

> **NOTA**
>
> O intervalo foi selecionado (`wdRng.Select`) para uma visualização mais fácil. Não é necessário que o intervalo esteja selecionado para ser manipulado. Por exemplo, para excluir o intervalo, faça isto:
>
> ```
> wdRng.Delete
> ```

A primeira posição de caractere em um documento é sempre zero e a última é equivalente ao número de caracteres no documento.

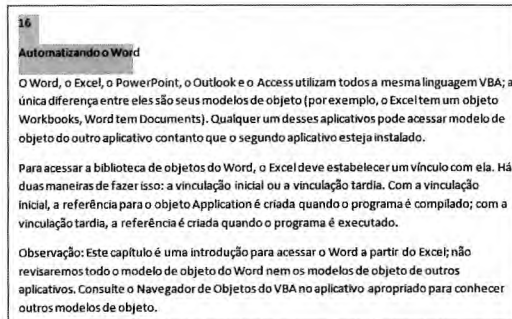
16

Automatizando o Word

O Word, o Excel, o PowerPoint, o Outlook e o Access utilizam todos a mesma linguagem VBA; a única diferença entre eles são seus modelos de objeto (por exemplo, o Excel tem um objeto Workbooks, Word tem Documents). Qualquer um desses aplicativos pode acessar modelo de objeto do outro aplicativo contanto que o segundo aplicativo esteja instalado.

Para acessar a biblioteca de objetos do Word, o Excel deve estabelecer um vínculo com ela. Há duas maneiras de fazer isso: a vinculação inicial ou a vinculação tardia. Com a vinculação inicial, a referência para o objeto Application é criada quando o programa é compilado; com a vinculação tardia, a referência é criada quando o programa é executado.

Observação: Este capítulo é uma introdução para acessar o Word a partir do Excel; não revisaremos todo o modelo de objeto do Word nem os modelos de objeto de outros aplicativos. Consulte o Navegador de Objetos do VBA no aplicativo apropriado para conhecer outros modelos de objeto.

Figura 18.7
O objeto `Range` seleciona tudo em seu caminho.

O objeto `Range` também seleciona parágrafos. O exemplo a seguir copia o terceiro parágrafo no documento ativo e o cola no Excel. Dependendo de como a colagem é feita, o texto pode ser colado em uma caixa de texto (veja Figura 18.8) ou em uma célula (veja Figura 18.9):

```
Sub SelectSentence()
Dim wdApp As Word.Application
Dim wdRng As Word.Range

Set wdApp = GetObject(, "Word.Application")

With wdApp.ActiveDocument
    If .Paragraphs.Count >= 3 Then
        Set wdRng = .Paragraphs(3).Range
        wdRng.Copy
    End If
End With

'Esta linha cola o texto copiado em uma caixa de texto porque esse é o método PasteSpecial padrão para texto do Word
Worksheets("Sheet2").PasteSpecial

'Esta linha cola o texto copiado na célula A1
Worksheets("Sheet2").Paste Destination:=Worksheets("Sheet2").Range("A1")

Set wdApp = Nothing
Set wdRng = Nothing
End Sub
```

Figura 18.8
Cole texto do Word em uma caixa de texto do Excel.

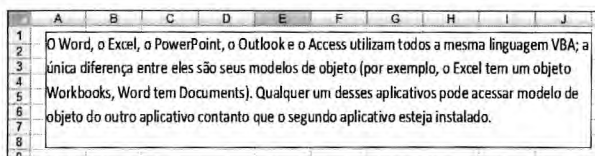
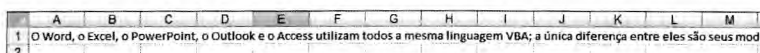



Figura 18.9
Cole texto do Word em uma célula do Excel.

Formate um intervalo

Depois de um intervalo ser selecionado, é possível formatá-lo (veja Figura 18.10). O programa a seguir faz loop por todos os parágrafos do documento ativo e aplica negrito à primeira palavra de cada parágrafo:

```
Sub ChangeFormat()
Dim wdApp As Word.Application
Dim wdRng As Word.Range
Dim count As Integer

Set wdApp = GetObject(, "Word.Application")

With wdApp.ActiveDocument
    For count = 1 To .Paragraphs.count
        Set wdRng = .Paragraphs(count).Range
        With wdRng
```

```
            .Words(1).Font.Bold = True
        End With
    Next count
End With

Set wdApp = Nothing
Set wdRng = Nothing
End Sub
```

Figura 18.10
Formate a primeira palavra de cada parágrafo em um documento.

O Word, o Excel, o PowerPoint, o Outlook e o Access utilizam todos a mesma linguagem VBA; a única diferença entre eles são seus modelos de objeto (por exemplo, o Excel tem um objeto Workbooks, Word tem Documents). Qualquer um desses aplicativos pode acessar modelo de objeto do outro aplicativo contanto que o segundo aplicativo esteja instalado.

Para acessar a biblioteca de objetos do Word, o Excel deve estabelecer um vínculo com ela. Há duas maneiras de fazer isso: a vinculação inicial ou a vinculação tardia. Com a vinculação inicial, a referência para o objeto Application é criada quando o programa é compilado; com a vinculação tardia, a referência é criada quando o programa é executado.

Observação: Este capítulo é uma introdução para acessar o Word a partir do Excel; não revisaremos todo o modelo de objeto do Word nem os modelos de objeto de outros aplicativos. Consulte o Navegador de Objetos do VBA no aplicativo apropriado para conhecer outros modelos de objeto.

Uma maneira rápida de alterar a formatação de parágrafos inteiros é alterar o estilo (veja Figura 18.11 e 18.12). O programa a seguir localiza o parágrafo com o estilo 'Sem estilo' e muda para o estilo 'HA':

```
Sub ChangeStyle()
Dim wdApp As Word.Application
Dim wdRng As Word.Range
Dim count As Integer

Set wdApp = GetObject(, "Word.Application")

With wdApp.ActiveDocument
    For count = 1 To .Paragraphs.count
        Set wdRng = .Paragraphs(count).Range
        With wdRng
            If .Style = "NO" Then
                .Style = "HA"
            End If
        End With
    Next count
End With

Set wdApp = Nothing
Set wdRng = Nothing
End Sub
```

Figura 18.11
Antes: um parágrafo com o estilo 'Sem estilo' precisa ser mudado para o estilo 'HA'.

Observação: Este capítulo é uma introdução para acessar o Word a partir do Excel; não revisaremos todo o modelo de objeto do Word nem os modelos de objeto de outros aplicativos. Consulte o Navegador de Objetos do VBA no aplicativo apropriado para conhecer outros modelos de objeto.

Figura 18.12
Depois: aplique estilos com código para mudar a formatação do parágrafo rapidamente.

Observação: Este capítulo é uma introdução para acessar o Word a partir do Excel; não revisaremos todo o modelo de objeto do Word nem os modelos de objeto de outros aplicativos. Consulte o Navegador de Objetos do VBA no aplicativo apropriado para conhecer outros modelos de objeto.

Indicadores

Os indicadores são membros dos objetos `Document`, `Selection` e `Range`. Eles ajudam a navegar pelo Word. Em vez de escolher palavras, frases ou parágrafos, use indicadores para manipular rapidamente as seções de um documento.

> **NOTA** Os indicadores não precisam ser existentes; podem ser criados por meio de código.

Os indicadores aparecem como barras I acinzentadas em documentos do Word. No Word, clique no botão Microsoft Office e vá para Opções do Word, Avançado, Mostrar Conteúdo do Documento para ativar o indicador (veja Figura 18.13).

Depois de configurar o indicador em um documento, você pode usá-lo para mover-se rapidamente em um intervalo. O código a seguir insere o texto automaticamente, depois que quatro indicadores foram previamente configurados no documento. A Figura 18.14 mostra os resultados.

```
Sub UseBookmarks()
Dim myArray()
Dim wdBkmk As String

Dim wdApp As Word.Application
Dim wdRng As Word.Range

myArray = Array("Para", "AC", "De", "Assunto")
Set wdApp = GetObject(, "Word.Application")

Set wdRng = wdApp.ActiveDocument.Bookmarks(myArray(0)).Range
wdRng.InsertBefore ("Bill Jelen")
Set wdRng = wdApp.ActiveDocument.Bookmarks(myArray(1)).Range
wdRng.InsertBefore ("Tracy Syrstad")
Set wdRng = wdApp.ActiveDocument.Bookmarks(myArray(2)).Range
wdRng.InsertBefore ("MrExcel")
Set wdRng = wdApp.ActiveDocument.Bookmarks(myArray(3)).Range
wdRng.InsertBefore ("Vendas de frutas")

Set wdApp = Nothing
Set wdRng = Nothing
End Sub
```

Figura 18.13
Ative indicadores para localizá-los em um documento.

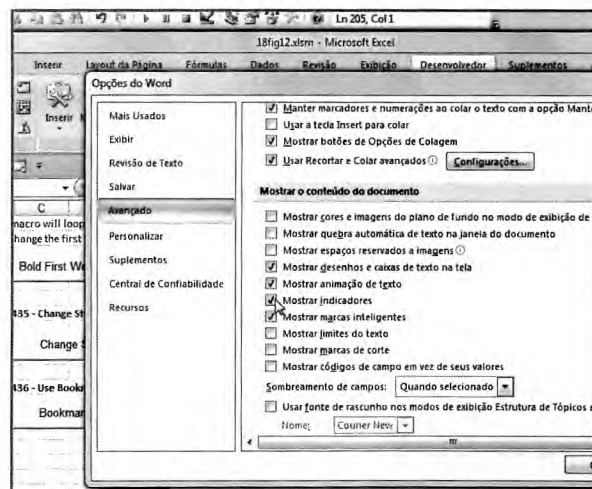
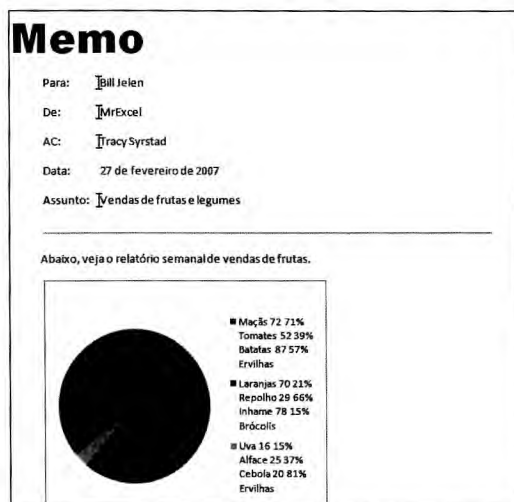



Figura 18.14
Utilize indicadores para inserir texto rapidamente em um documento do Word.

Os indicadores também podem ser utilizados como marcadores a serem usados em gráficos criados no Excel. O código a seguir vincula um gráfico do Excel (veja Figura 18.15) a um memorando:

```
Sub CreateMemo()
Dim myArray()
Dim wdBkmk As String

Dim wdApp As Word.Application
Dim wdRng As Word.Range
```

```
myArray = Array("Para", "AC", "De", "Assunto", "Chart")
Set wdApp = GetObject(, "Word.Application")

Set wdRng = wdApp.ActiveDocument.Bookmarks(myArray(0)).Range
wdRng.InsertBefore ("Bill Jelen")
Set wdRng = wdApp.ActiveDocument.Bookmarks(myArray(1)).Range
wdRng.InsertBefore ("Tracy Syrstad")
Set wdRng = wdApp.ActiveDocument.Bookmarks(myArray(2)).Range
wdRng.InsertBefore ("MrExcel")
Set wdRng = wdApp.ActiveDocument.Bookmarks(myArray(3)).Range
wdRng.InsertBefore ("Vendas de frutas & legumes")

Set wdRng = wdApp.ActiveDocument.Bookmarks(myArray(4)).Range
ActiveSheet.ChartObjects("Chart 1").Copy
wdRng.PasteAndFormat Type:=wdPasteOLEObject

wdApp.Activate

Set wdApp = Nothing
Set wdRng = Nothing
End Sub
```

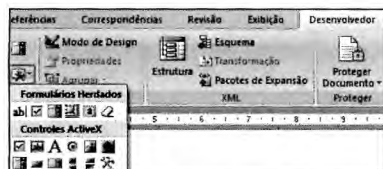
Figura 18.15
Utilize indicadores para inserir gráficos em um documento do Word.

Controlando campos de formulário do Word

Você viu como modificar um documento inserindo gráficos e texto, modificando formatação e excluindo texto. Mas um documento pode conter outros itens, como controles, que também podem ser modificados.

Para o exemplo a seguir, criei um modelo (template) composto de texto, indicadores e caixas de seleção Campo de Formulário. (Veja a nota depois do parágrafo a seguir para obter informações sobre onde os Campos de Formulário estão ocultos no Word.)

Os indicadores são colocados depois dos campos Nome e Data. As caixas de seleção foram renomeadas (clique com o botão direito do mouse na caixa de seleção, selecione Propriedades e digite um novo nome no campo Indicadores) para atender aos meus propósitos, como chkFundosInvest em vez de Checkbox5. Salve o modelo.

> **NOTA** Os *Campos de* Formulário do Word estão localizados na seção Controles da guia Desenvolvedor, em Ferramentas Herdadas, como mostrado na Figura 18.16.

Figura 18.16
Você pode utilizar os Campos de Formulário localizados em Ferramentas Herdadas para adicionar caixas de seleção a um documento.

Configurei o questionário no Excel, *permitindo* ao usuário inserir texto livre em B1 e B2, mas com validação de dados em B3 e B5:B8, como mostrado na Figura 18.17.

	A	B
1	Nome	Mary Beth Jacobson
2	Data	27/08/2008
3	Você é um novo cliente?	Yes
4	Você está interessado nas seguintes opções:	
5	Fundos de investimento	Yes
6	CDB	No
7	Ações	Yes
8	Poupança	Yes
9		Yes
10		No

Figura 18.17
Crie uma planilha Excel para coletar seus dados.

O código entra em um módulo-padrão. O nome e a data vão diretamente no documento. As caixas de seleção utilizam a lógica para verificar se o usuário selecionou 'Yes' ou 'No' para confirmar se a caixa de seleção correspondente deve ou não ser marcada. A Figura 18.18 mostra um documento de exemplo pronto.

```
Sub FillOutWordForm()

Dim TemplatePath As String
Dim wdApp As Object
Dim wdDoc As Object

'Abre o modelo em uma nova instância do Word
TemplatePath = ThisWorkbook.Path & "\Word Example.dotx"
Set wdApp = CreateObject("Word.Application")
Set wdDoc = wdApp.documents.Add(Template:=TemplatePath)

'Posiciona os valores de texto no documento
With wdApp.ActiveDocument
    .Bookmarks("Nome").Range.InsertBefore Range("B1").Text
    .Bookmarks("Data").Range.InsertBefore Range("B2").Text
End With

'Utilizando lógica básica, seleciona o objeto de formulário correto
If Range("B3").Value = "Yes" Then
    wdDoc.formfields("chkCustYes").CheckBox.Value = True
Else
    wdDoc.formfields("chkCustNo").CheckBox.Value = True
End If

With wdDoc
    If Range("B5").Value = "Yes" Then .Formfields("chkFundosInvest").CheckBox.Value = True
    If Range("B6").Value = "Yes" Then .Formfields("chkCDB").CheckBox.Value = True
    If Range("B7").Value = "Yes" Then .Formfields("chkAcoes"). _
CheckBox.Value = True
    If Range("B7").Value = "Yes" Then .Formfields("chkPoupanca"). _
CheckBox.Value = True
End With

wdApp.Visible = True

ExitSub:

    Set wdDoc = Nothing
    Set wdApp = Nothing

End Sub
```

Figura 18.18
O Excel pode controlar os campos de formulário do Word.

Próximos passos

No Capítulo 19, "Matrizes", você aprenderá a utilizar matrizes multidimensionais. Ler dados em uma matriz multidimensional, realizar cálculos na matriz e, então, reescrevê-la para um intervalo pode acelerar significativamente suas macros.

Matrizes

Matriz (ou array) é um tipo de variável que pode ser utilizada para armazenar mais de uma parte de dados. Por exemplo, se você tiver de trabalhar com o nome e o endereço de um cliente, a primeira coisa em que poderia pensar seria em atribuir uma variável para o nome e outra para o endereço do cliente. Em vez disso, considere a possibilidade de usar uma matriz, que pode armazenar os dois fragmentos de informações — e não apenas de um cliente, mas de centenas deles.

Declare uma matriz

Declare uma matriz colocando parênteses depois do nome dela. Os parênteses contêm o número de elementos na matriz:

```
Dim myArray (2)
```

Isso cria uma matriz, `myArray`, que contém três elementos. Por que três? Porque, por padrão, a contagem de índice inicia em zero:

```
myArray(0) = 10
myArray(1) = 20
myArray(2) = 30
```

Se a contagem de índice tiver de iniciar em 1, utilize `Option Base 1`. Isso força a contagem iniciar em 1. A instrução `Option Base` é colocada na seção de declarações do módulo:

```
Option Base 1
Dim myArray(2)
```

Agora isso força a matriz a ter apenas dois elementos.

Você também pode criar uma matriz independente da instrução `Option Base` declarando seu limite inferior:

```
Dim myArray (1 to 10)
Dim BigArray (100 to 200)
```

Toda matriz tem um limite inferior (`Lbound`) e um superior (`Ubound`). Ao declarar `Dim myArray (2)`, você declara o limite superior e permite que a base de opção declare o limite inferior. Declarando `Dim myArray (1 to 10)`, você declara o limite inferior, 1, e o limite superior, 10.

Matrizes multidimensionais

As matrizes dos quais acabamos de falar são consideradas matrizes unidimensionais — um único número designa o local de um elemento na matriz. A matriz é como uma única linha de dados mas, como pode haver apenas uma linha, não é preciso se preocupar com o número de linhas — somente com o número de colunas. Por exemplo, para recuperar o segundo elemento (`Option Base 0`), use `myArray (1)`.

Em alguns casos, uma única dimensão não é suficiente. É aqui que as matrizes multidimensionais entram em cena. Enquanto uma matriz dimensional é uma única linha de dados, uma matriz multidimensional contém linhas e colunas.

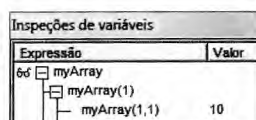
> **NOTA** Outra palavra para matriz é *array*, que é o que uma planilha significa. O objeto `Cells` referencia *elementos* de uma planilha — e uma célula consiste de uma linha e uma coluna. Você tem utilizado matrizes esse tempo todo!

Para declarar outra dimensão a uma matriz, adicione outro argumento. A linha a seguir cria uma matriz de 10 linhas e 20 colunas:

```
Dim myArray (1 To 10, 1 To 20)
```

Isso coloca valores nas duas primeiras colunas da primeira linha, como mostrado na Figura 19.1:

```
myArray (1,1) = 10
myArray (1,2) = 20
```

Figura 19.1
A janela Inspeções do Editor do VB mostra a primeira 'linha' da matriz sendo preenchida com as linhas anteriores de código.

Isso coloca valores nas duas primeiras colunas da segunda linha:

```
myArray (2,1) = 20
myArray (2,2) = 40
```

E assim por diante. Isso, evidentemente, é muito demorado e poderia ter uma grande quantidade de linhas de código. Há outras formas de preencher uma matriz, que serão discutidas na próxima seção.

Preencha uma matriz

Agora que você sabe declarar uma matriz, precisa preenchê-la. Já mostramos um método antes — inserir individualmente um valor em cada elemento da matriz. Há uma maneira mais rápida, como mostrado no próximo código de exemplo e na Figura 19.2:

```
Option Base 1

Sub ColumnHeaders()
Dim myArray As Variant
Dim myCount As Integer

'Preenche a matriz
myArray = Array("Nome", "Endereço", "Telefone", "E-mail")

'Esvazia a matriz
With Worksheets("Sheet2")
    For myCount = 1 To UBound(myArray)
        .Cells(1, myCount).Value = myArray(myCount)
    Next myCount
End With
End Sub
```

	A	B	C	D
1	Nome	Endereço	Telefone	Email
2				
3				
4				

Figura 19.2
Utilize uma matriz para criar cabeçalhos de coluna rapidamente.

Lembre-se de que as variáveis `Variant` podem armazenar qualquer tipo de informação. Para construir a matriz rapidamente, crie uma variável de tipo `Variant`, que pode ser tratada como uma matriz. Quando os dados são colocados na variante, eles são forçados a assumir as propriedades de uma matriz.

Mas, e se as informações necessárias à matriz já estiverem na planilha? Use o seguinte código para preencher uma matriz rapidamente.

```
Dim myArray As Variant
myArray = Worksheets("Sheet1").Range("B2:C17")
```

Embora esses dois métodos sejam rápidos e fáceis, nem sempre são os adequados à situação. E se você precisar de uma linha sim outra não na matriz? O código para fazer isso é o seguinte (veja Figura 19.3):

```
Sub EveryOtherRow()
Dim myArray(1 To 8, 1 To 2)
Dim i As Integer, j As Integer, myCount As Integer

'Preenche a matriz com linhas alternadamente
For i = 1 To 8
    For j = 1 To 2
        myArray(i, j) = Worksheets("Sheet1").Cells(i * 2, j + 1).Value
    Next j
Next i

'Esvazia a matriz
For myCount = LBound(myArray) To UBound(myArray)
    Worksheets("Sheet1").Cells(myCount * 2, 4) =WorksheetFunction.Sum(myArray(myCount, 1), myArray(myCount, 2))
Next myCount
End Sub
```

Figura 19.3
Preencha a matriz apenas com os dados necessários.

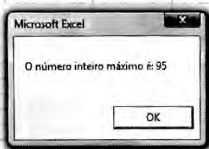
	A	B	C	D
1		Dez '06	Jan '07	Soma
2	Maçãs	45	0	45
3	Laranjas	12	10	
4	Uvas	86	12	98
5	Limões	15	15	
6	Tomates	58	24	82
7	Repolho	24	26	
8	Alface	31	29	60

Esvazie uma matriz

Depois de uma matriz ser preenchida, os dados precisam ser recuperados. Mas, antes de fazer isso, você pode manipulá-los ou retornar informações sobre eles, como o número inteiro máximo, conforme mostrado no próximo código (veja Figura 19.4):

```
Sub QuickFillMax()
Dim myArray As Variant

myArray = Worksheets("Sheet1").Range("B2:C17")
MsgBox "O número inteiro máximo é 95: " & WorksheetFunction.Max(myArray)

End Sub
```

Figura 19.4
Retorne a variável `Max` em uma matriz.

	A	B	C
1		Dez '06	Jan '07
2	Maçãs	45	0
3	Laranjas	12	10
4	Uvas	86	12
5	Limões	15	15
6	Tomates	58	24
7	Repolho	24	26
8	Alface	31	29
9	Pimenta	0	31
10	Batatas	10	45
11	Inhame	61	46
12	Cebolas	26	58
13	Alho	29	61
14	Ervilhas	46	64
15	Brócolis	64	79
16	Ervilhas	79	86
17	Cenouras	95	95

Os dados também podem ser manipulados à medida que são retornados à planilha. No exemplo a seguir, `Lbound` e `Ubound` são utilizados com um loop `For` pelos elementos da matriz e média para cada conjunto. O resultado é colocado na planilha em uma nova coluna (veja Figura 19.5):

> **NOTA** `MyCount+1` é utilizado para recolocar os resultados na planilha porque `Lbound` é 1 e os dados iniciam na linha 2.

```
Sub QuickFillAverage()
Dim myArray As Variant
Dim myCount As Integer
'Preenche a matriz
myArray = Worksheets("Sheet1").Range("B2:C17")

'Calcula a média dos dados na matriz assim que são colocados na planilha
For myCount = LBound(myArray) To UBound(myArray)
    Worksheets("Sheet1").Cells(myCount + 1, 5).Value =WorksheetFunction.Average(myArray(myCount, 1),
myArray(myCount, 2))
Next myCount

End Sub
```

Figura 19.5
Os cálculos podem ser feitos nos dados quando são retornados à planilha.

	A	B	C	D	E
1		Dez '06	Jan '07	Soma	Média
2	Maçãs	45	0	45	22,5
3	Laranjas	12	10		11
4	Uvas	86	12	98	49
5	Limões	15	15		15
6	Tomates	58	24	82	41
7	Repolho	24	26		25
8	Alface	31	29	60	30
9	Pimenta	0	31		15,5
10	Batatas	10	45	55	27,5
11	Inhame	61	46		53,5
12	Cebolas	26	58	84	42
13	Alho	29	61		45
14	Ervilhas	46	64	110	55
15	Brócolis	64	79		71,5
16	Ervilhas	79	86	165	82,5
17	Cenouras	95	95		95

As matrizes facilitam a manipulação de dados, mas isso é tudo?

Certo, então as matrizes podem tornar mais fácil a manipulação de dados e a obtenção de informações sobre eles — mas é só para isso que elas servem? Não, as matrizes são tão poderosas porque podem, realmente, tornar o código mais rápido!

Em geral, se houver cálculos de média de colunas de dados a serem feitos, como no exemplo anterior, a primeira coisa em que você poderia pensar é o seguinte:

```
Sub SlowAverage()
Dim myCount As Integer, LastRow As Integer

LastRow = Worksheets("Sheet1").Cells(Worksheets("Sheet1").Rows.Count, 1).End(xlUp).Row

For myCount = 2 To LastRow
    With Worksheets("Sheet1")
        .Cells(myCount, 6).Value =WorksheetFunction.Average(Cells(myCount, 2), Cells(myCount, 3))
    End With
Next myCount

End Sub
```

Embora isso funcione bem, o programa tem de examinar individualmente cada linha da planilha, obter os dados, fazer o cálculo e colocá-los na coluna correta. Não seria mais fácil obter todos os dados de uma vez e, depois, fazer o cálculo e colocar os dados novamente na planilha? Além disso, com a versão mais lenta do código, você precisa saber quais colunas na planilha manipular (colunas 2 e 3, em nosso exemplo). Com uma matriz, você só precisa definir que elemento da matriz irá manipular.

Para tornar isso ainda mais útil, em vez de utilizar um intervalo de endereço para preencher a matriz, você poderia utilizar um intervalo nomeado. Com um intervalo nomeado em uma matriz, realmente não importa onde ele está localizado na planilha.

Em vez de

```
myArray = Range("B2:C17")
```

utilize isto:

```
myArray = Range("myData")
```

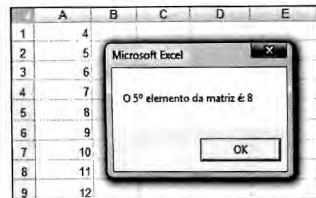
Embora com o método lento você precise saber onde está `myData` a fim de retornar as colunas corretas, com uma matriz, você só precisa saber que quer a primeira e segunda colunas.

DICA

Torne a matriz ainda mais rápida! Tecnicamente, se você colocar uma coluna de dados em uma matriz, ela será uma matriz bidimensional. Se quiser processá-la, você deverá processar a linha e a coluna.

Você pode processar a coluna mais facilmente se ela tiver apenas uma linha, contanto que esta não exceda 16.384 colunas. Utilize a função `Transpose` para transformar uma coluna em uma linha (veja Figura 19.6).

```
Sub TransposeArray()
Dim myArray As Variant

    myArray = WorksheetFunction.Transpose(Range("myTran"))

    'retorna o 5º elemento da matriz
    MsgBox "O 5º elemento da matriz é: " & myArray(5)
End Sub
```

Figura 19.6
Utilize a função `Transpose` para transformar uma matriz bidimensional em uma matriz dimensional.

Matrizes dinâmicas

Nem sempre você pode saber qual será a extensão da matriz de que precisará. Seria possível criar uma matriz baseada no tamanho máximo que ela precisaria ter, mas isso é um desperdício de memória. E se a matriz tiver de ser ainda maior? Nesse caso, você pode utilizar uma matriz dinâmica.

Uma matriz dinâmica é aquela que não tem um tamanho definido. Declare a matriz, mas deixe os parênteses vazios:

```
Dim myArray ()
```

Mais tarde, quando o programa for utilizar a matriz, utilize `Redim` para configurar o tamanho dela. O programa seguinte, que retorna os nomes de todas as planilhas na pasta de trabalho, cria uma matriz ilimitada, mas, em seguida, define o limite superior, depois de saber quantas planilhas estão na pasta de trabalho:

```
Option Base 1
Sub MySheets()
Dim myArray() As String
Dim myCount As Integer, NumShts As Integer

NumShts = ActiveWorkbook.Worksheets.Count

'Dimensiona a matriz
ReDim myArray(1 To NumShts)

For myCount = 1 To NumShts
    myArray(myCount) = ActiveWorkbook.Sheets(myCount).Name
Next myCount

End Sub
```

Utilizar `Redim` reinicializa a matriz; então, se fosse utilizá-la muitas vezes, como em um loop, você perderia todos os dados que ela armazena. Utilize `Preserve` para impedir que isso ocorra.

O exemplo seguinte procura todos os arquivos Excel em um diretório e coloca os resultados em uma matriz. Como não sabemos quantos serão os arquivos até examiná-los, não podemos dimensionar a matriz antes que o programa seja executado:

```
Sub XLFiles()
Dim FName As String
Dim arNames() As String
Dim myCount As Integer

FName = Dir("C:\Contracting Files\Excel VBA 2007 by Jelen & Syrstad\*.xls*")
Do Until FName = ""
    myCount = myCount + 1
    ReDim Preserve arNames(1 To myCount)
    arNames(myCount) = FName
    FName = Dir
Loop

End Sub
```

ATENÇÃO

Utilizar `Preserve` com grandes quantidades de dados em um loop diminui a velocidade do programa. Se possível, utilize código para descobrir o tamanho máximo da matriz.

Passando uma matriz

Assim como as strings, os inteiros e outras variáveis, as matrizes podem ser passadas para outros procedimentos. Isso torna o código mais eficiente e mais fácil de ser lido. O procedimento `Sub PassAnArray` passa a matriz `myArray` para a função `RegionSales`. Os dados na matriz são somados para a região especificada e o resultado é retornado para a sub.

```
Sub PassAnArray()
Dim myArray() As Variant
Dim myRegion As String

myArray = Range("mySalesData")
myRegion = InputBox("Insira Região - Central, Leste, Oeste")
MsgBox myRegion & " As vendas são: " & Format(RegionSales(myArray,myRegion), "$#,#00.00")

End Sub

Function RegionSales(ByRef BigArray As Variant, sRegion As String) As Long
Dim myCount As Integer

RegionSales = 0
For myCount = LBound(BigArray) To UBound(BigArray)
    If BigArray(myCount, 1) = sRegion Then
        RegionSales = BigArray(myCount, 6) + RegionSales
    End If
Next myCount

End Function
```

Próximos passos

As matrizes são um tipo de variável utilizado para armazenar mais de um fragmento de dados. O Capítulo 20, "Processamento de arquivo de texto", discute como importar o conteúdo de um arquivo de texto e gravá-lo em um arquivo de texto. Saber gravar em arquivo de texto é útil quando você tiver de gravar dados para serem lidos em outro sistema ou quando tiver de criar arquivos HTML.

Processamento de arquivo de texto

20

Apesar da promessa no Capítulo 17, "XML no Excel 2007", de que o XML será o próximo grande formato de arquivo do futuro, ainda hoje temos uma grande quantidade de arquivos no formato de arquivo CSV ou TXT.

O VBA facilita tanto a leitura como a gravação de arquivos de texto. Este capítulo explica como importar um arquivo de texto e gravar em um arquivo de texto. Saber gravar em um arquivo de texto revela-se útil quando você precisa gravar dados para ser lido em outro sistema ou até mesmo se precisar produzir arquivos de HTML.

Importando a partir de arquivos de texto

Há dois cenários básicos ao ler arquivos de texto. Se o arquivo contiver menos de 1.048.576 registros, é fácil importá-lo utilizando o método `Workbooks.OpenText`. Se o arquivo contiver mais de 1.048.576 registros, ele terá de ser lido um registro por vez.

Importando arquivo de texto com menos de 1.084.576 linhas

Os arquivos de texto em geral são fornecidos em um de dois formatos. No primeiro formato, os campos em cada registro são separados por algum delimitador, como uma vírgula, barra vertical ou tabulação. No segundo formato, cada campo aceita um determinado número de posições de caractere. Isso é chamado de arquivo de largura fixa e foi muito popular na época do COBOL.

O Excel pode importar qualquer tipo de arquivo facilmente. Você pode abrir ambos os tipos utilizando o método `OpenText`. Em ambos os casos, é melhor gravar o processo de abrir o arquivo e utilizar o trecho registrado de código.

Abrindo um arquivo de largura fixa

A Figura 20.1 mostra um arquivo de texto, no qual cada campo ocupa certa quantidade de espaço no registro. Escrever o código para abrir esse tipo de arquivo é um pouco difícil, porque é preciso especificar o comprimento de cada campo. Em minha coleção de antigüidades, ainda tenho a régua de metal utilizada pelos programadores COBOL para medir o número de caracteres em um campo impresso em uma impressora greenbar. Você poderia, na teoria, alterar

vendas.prn - Bloco de notas

Arquivo Editar Formatar Exibir Ajuda

```
Regiao  Produto Data     Cliente Quantidade Receita  CustoLucro
Leste   XYZ     7/24/200QRS INC.     1000         228101022012590
Central DEF     7/25/200JKL, CO       100          2257  984 1273
Leste   ABC     7/25/200JKL, CO       500         10245 4235 6010
Central XYZ     7/26/200WXY, CO       500         11240 5110 6130
Leste   XYZ     7/27/200FGH, CO       400          9152 4088 5064
Central XYZ     7/27/200WXY, CO       400          9204 4088 5116
Leste   DEF     7/27/200RST INC       800         18552 787210680
Central ABC     7/28/200EFG S.A.      400          6860 3388 3472
Leste   DEF     7/30/200UVW, INC     1000         21730 984011890
Oeste   XYZ     7/30/200FGH, CO       600         13806 6132 7674
Leste   ABC     7/30/200FGH LTD.      400          8456 3388 5068
Leste   XYZ     8/1/2004MNO S.A.      900         21015 919811817
Central ABC     8/1/2004FGH LTD.      800         16416 6776 9640
Central XYZ     8/2/2004OPQ, INC      900         21438 919812240
Leste   XYZ     8/2/2004WXY, CO       900         21465 919812267
Oeste   XYZ     8/4/2004WXY, CO       400          9144 4088 5056
Central ABC     8/4/2004EFG S.A.      300          6267 2541 3726
Central ABC     8/6/2004TUV GMBH      100          1740  847  893
Oeste   ABC     8/6/2004OPQ, INC     1000         19110 847010640
Leste   XYZ     8/6/2004TUV GMBH      100          2401 1022 1379
Leste   ABC     8/7/2004JKL, CO       500          9345 4235 5110
Leste   ABC     8/8/2004LMN PTY       600         11628 5082 6546
Central XYZ     8/8/2004OPQ, INC      900         21888 919812690
Leste   XYZ     8/9/2004DEF, LLC      300          5961 2952 3009
Oeste   DEF     8/11/200JKL, CO       100          2042  984 1058
Central ABC     8/12/200LMN PTY       900         17505 7623 9882
Oeste   ABC     8/13/200RST INC.      200          3552 1694 1858
Leste   ABC     8/13/200LMN LTD.      800         14440 6776 7664
Oeste   XYZ     8/13/200DEF, LLC      300          7032 2952 4080
```

Figura 20.1
Esse arquivo é um arquivo delimitado por espaço ou de largura fixa. Como você deve especificar o comprimento exato de cada campo no arquivo, abrir esse arquivo é relativamente complicado.

a fonte de seu arquivo para uma fonte monoespaçada e utilizar esse mesmo método. Mas utilizar o programa de gravação de macros é um método ligeiramente mais moderno.

Desative o programa de gravação de macros selecionando Gravar Macro na guia Desenvolvedor da Faixa. No menu Arquivo, selecione Abrir. Mude de Arquivos de Texto para Todos os Arquivos e localize o arquivo de texto.

No Passo 1 do Assistente de Importação de Texto, especifique que os dados têm Largura Fixa e clique em Avançar.

O Excel vê os dados e tenta descobrir onde cada campo inicia e acaba. A Figura 20.2 mostra a suposição do Excel nesse arquivo em particular. Como o campo Data também está ao lado do campo Cliente, o Excel não conseguiu desenhar aquela linha.

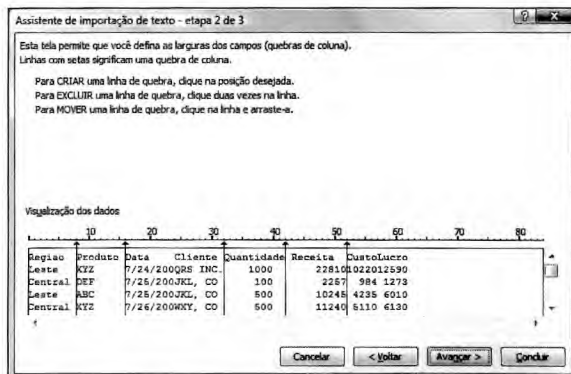
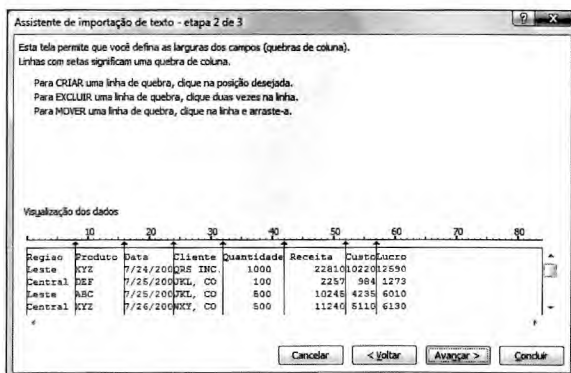
Figura 20.2
O Excel supõe onde cada campo começa. Nesse caso, ele perdeu dois campos e provavelmente não deixou espaço suficiente para um nome de produto mais longo.

Para adicionar um novo indicador de campo do assistente no Passo 2, apenas clique no lugar apropriado na janela Visualização de Dados. Se você clicar na coluna errada, clique na linha e arraste-a para o lugar certo. Se o Excel colocar inadvertidamente uma linha de campo extra, dê um clique duplo na linha para removê-la. A Figura 20.3 mostra a visualização de dados depois que as alterações apropriadas foram feitas. Note a pequena régua acima dos dados. Quando você clica para adicionar um marcador de campo, na realidade o Excel está realizando a tediosa tarefa de descobrir que o campo Cliente começa na posição 25 e tem comprimento 11.

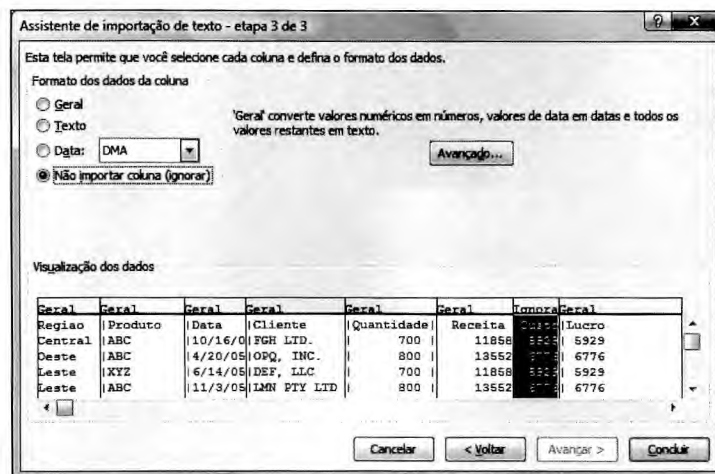
Figura 20.3
Depois de adicionar dois novos marcadores de campo e mover o marcador entre Produto e Data para o lugar certo, o Excel pode construir o código que nos dá uma idéia da posição e comprimento inicial de cada campo.

No Passo 3 do assistente, o Excel sempre assume que cada campo está no formato Geral.

Mude o formato de qualquer campo que requeira tratamento especial. Clique na coluna e escolha o formato apropriado na seção Formato de Dados de Coluna da caixa de diálogo. A Figura 20.4 mostra as seleções para esse arquivo.

Figura 20.4
Aqui, indiquei que a terceira coluna é uma data e que eu não quero importar as colunas Custo e Lucro. No Brasil, assim como na Itália, utilizam-se pontos como separador de milhares. Então, você precisará clicar o botão Avançado para configurar as opções para tratar isso.

Se você tiver campos de data, clique no título acima dessa coluna e mude a escolha de formato de dados de coluna para data. Se tiver um arquivo com datas no formato ano–mês–dia ou dia–mês–ano, selecione a lista suspensa ao lado da data e escolha a seqüência de data adequada.

Se preferir pular alguns campos, clique nessa coluna e selecione Não Importar Coluna (Ignorar) na seleção Formato dos Dados da Coluna. Há algumas ocasiões em que isso é útil. Se o arquivo inclui dados confidenciais que você não quer mostrar ao cliente, é possível deixá-los fora da importação. Por exemplo, se esse relatório for para um cliente e eu não quiser mostrar-lhe o custo das mercadorias vendidas ou lucros, posso escolher pular esses campos na importação. Além disso, ocasionalmente você encontrará um arquivo de texto que é tanto de largura fixa como delimitado por caracteres, como os de barra vertical. Definir as colunas formadas pelas barras verticais como 'não importar' é uma excelente maneira de eliminar esses caracteres, como mostrado na Figura 20.5.

Figura 20.5
Esse arquivo é tanto de largura fixa como delimitado pelo caractere de barra vertical. A configuração Não Importar Coluna permite eliminar os caracteres de barra vertical do arquivo.

Se você tiver campos de texto que contêm caracteres alfabéticos, escolha o formato Geral. A única vez em que se deve escolher o formato Texto é quando existir um campo numérico que tenha de ser importado explicitamente como texto. Um exemplo disso é um número de conta com zeros à esquerda ou uma coluna de CEP. Para assegurar que o CEP 01234 não perderá o zero à esquerda, mude o campo para o formato Texto.

ATENÇÃO

Depois de importar um arquivo de texto e especificar que aquele campo é texto, o campo exibirá um comportamento aparentemente esquisito. Tente inserir uma nova linha e uma fórmula no meio de uma coluna importada como texto. Em vez de obter os resultados da fórmula, o Excel irá inserir a fórmula como texto. A solução é excluir a fórmula, formatar a coluna inteira como Geral e depois inserir a fórmula de novo.

Depois de abrir o arquivo, desative o programa de gravação de macros e examine o código gravado:

```
Workbooks.OpenText Filename:="C:\vendas.prn", Origin:=437, StartRow:=1, DataType:=xlFixedWidth, _
    FieldInfo:=Array(Array(0, 1), Array(8, 1), Array(17, 3), Array(25, 1), Array(36, 1), Array(46, 1), _
    Array(56, 9),  Array(61, 9)), TrailingMinusNumbers:=True
```

A parte de mais confusa desse código é o parâmetro `FieldInfo`. Você deve codificar uma matriz de arrays de dois elementos. Todo campo no arquivo obtém um array de dois elementos para identificar onde o campo inicia e o tipo de campo.

A posição inicial do campo é baseada em zero; como o campo Região está na primeira posição de caractere, sua posição inicial é listada como 0.

O tipo de campo é um código numérico. Se você estivesse codificando isso manualmente, utilizaria os nomes de constante `xlColumnDataType`; mas, por alguma razão, o programa de gravação de macros utiliza os equivalentes numéricos mais difíceis de entender.

Com a Tabela 20.1, você pode decodificar o significado dos arrays individuais no array `FieldInfo`. `Array(0, 1)` significa que esse campo inicia caracteres zeros da borda esquerda do arquivo e é um formato geral. `Array(8, 1)` indica que o próximo campo inicia oito caracteres da borda esquerda do arquivo e também é o formato Geral. `Array(17, 3)` indica que o próximo campo inicia 17 caracteres a partir da borda esquerda do arquivo e é um formato de data na seqüência mês–dia–ano.

Tabela 20.1 Valores `xlColumnDataType`

Valor	Constante	Utilizado para
1	xlGeneralFormat	Geral
2	xlTextFormat	Texto
3	xlMDYFormat	Data MDA
4	xlDMYFormat	Data DMA
5	xlYMDFormat	Data AMD
6	xlMYDFormat	Data MAD
7	xlDYMFormat	Data DAM
8	xlYDMFormat	Data ADM
9	xlSkipColumn	Ignorar coluna
10	xlEMDFormat	Data EMD

Como você pode ver, o parâmetro `FieldInfo` para arquivos de largura fixa é difícil de ser codificado e examinado. De fato, é uma situação em que é mais fácil gravar a macro e copiar o trecho de código.

ATENÇÃO

O parâmetro `xlTrailingMinusNumbers` era novo no Excel 2002. Se você tiver quaisquer clientes que ainda utilizem o Excel 97 ou 2000, tire o parâmetro que estiver gravado. O código executa bem sem o parâmetro nas versões mais novas, mas, se presente nas versões mais antigas, resultará em erro de compilação. Em minha experiência, essa é a principal causa de falha de código nas primeiras versões do Excel.

Abrindo um arquivo delimitado

A Figura 20.6 mostra um arquivo de texto em que cada campo é separado por vírgula. A tarefa principal ao abrir um arquivo desse tipo é informar o Excel de que o delimitador no arquivo é uma vírgula e, então, identificar qualquer processamento especial para cada campo. Nesse caso, queremos definitivamente identificar a terceira coluna como uma data no formato mm/dd/aaaa.

ATENÇÃO

Se você tentar registrar o processo de abrir um arquivo delimitado por vírgulas em que o nome de arquivo termina em .csv, o Excel registrará o método `Workbooks.Open` em vez de `Workbooks.OpenText`. Se você tiver de controlar a formatação de certas colunas, renomeie o arquivo para que ele tenha uma extensão .txt antes de gravar a macro.

Figura 20.6
Esse arquivo é delimitado por vírgula. Abri-lo envolve pedir para o Excel procurar uma vírgula como o delimitador e, então, identificar qualquer tratamento especial, como tratar a terceira coluna como uma data. Isso é muito mais fácil do que tratar arquivo de largura fixa.

vendas.csv - Bloco de notas

Arquivo Editar Formatar Exibir Ajuda

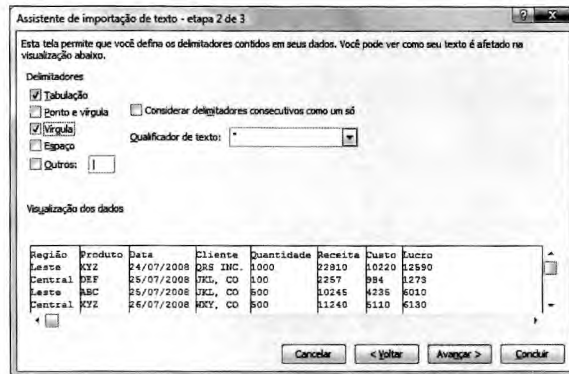
```
Região,Produto,Data,Cliente,Quantidade,Receita,Custo,Lucro
Leste,XYZ,24/07/2008,QRS INC.,1000,22810,10220,12590
Central,DEF,25/07/2008,"JKL, CO",100,2257,984,1273
Leste,ABC,25/07/2008,"JKL, CO",500,10245,4235,6010
Central,XYZ,26/07/2008,"WXY, CO",500,11240,5110,6130
Leste,XYZ,27/07/2008,"FGH, CO",400,9152,4088,5064
Central,XYZ,27/07/2008,"WXY, CO",400,9204,4088,5116
Leste,DEF,27/07/2008,RST INC.,800,18552,7872,10680
Central,ABC,28/07/2008,EFG S.A.,400,6860,3388,3472
Leste,DEF,30/07/2008,"UVW, INC.",1000,21730,9840,11890
Oeste,XYZ,30/07/2008,"FGH, CO",600,13806,6132,7674
Leste,ABC,30/07/2008,FGH LTD.,400,8456,3388,5068
Leste,XYZ,01/08/2008,MNO S.A.,900,21015,9198,11817
Central,ABC,01/08/2008,FGH LTD.,800,16416,6776,9640
Central,XYZ,02/08/2008,"OPQ, INC.",900,21438,9198,12240
Leste,XYZ,02/08/2008,"WXY, CO",900,21465,9198,12267
Oeste,XYZ,04/08/2008,"WXY, CO",400,9144,4088,5056
```

Ative o programa de gravação de macros e grave o processo de abrir o arquivo de texto. No Passo 1 do assistente, especifique que o arquivo é delimitado.

No Assistente de Importação de Texto — a Etapa 2 de 3, a visualização dos dados pode inicialmente parecer horrível. Isso porque o Excel assume, por padrão, que todo campo é separado por um caractere de tabulação (veja Figura 20.7).

Depois de desmarcar a caixa Tabulação e verificar a escolha do delimitador adequado (nesse caso, uma vírgula), a visualização dos dados na Etapa 2 parece perfeita, como mostrado na Figura 20.8.

Figura 20.7
Antes de você importar um arquivo de texto delimitado, a visualização de dados inicial parece *confusa*, porque o Excel está procurando caracteres de tabulação entre cada campo quando, de fato, uma vírgula é o delimitador nesse arquivo.

Figura 20.8
Depois de mudar o campo delimitador de uma tabulação para uma vírgula, a visualização de dados parece perfeita. Sem dúvida, isso é mais fácil que o incômodo processo da Etapa 2 para um arquivo de largura fixa.

A Etapa 3 do assistente é idêntica à Etapa 3 de um arquivo de largura fixa. Nesse caso, especifique que a terceira coluna tem um formato de data. Clique em Concluir e terá esse código no programa de gravação de macros:

```
Workbooks.OpenText Filename:="C:\vendas.txt", Origin:=437,StartRow:=1, DataType:=xlDelimited, _
    TextQualifier:=xlDoubleQuote,ConsecutiveDelimiter:=False, Tab:=False, Semicolon:=False, Comma:=True, _
    Space:=False, Other:=False, FieldInfo:=Array(Array(1, 1), Array(2, 1),Array(3, 3), Array(4, 1), _
    Array(5, 1), Array(6, 1), Array(7, 1), Array(8, 1)), TrailingMinusNumbers:=True
```

Embora esse código pareça mais longo, na verdade ele é muito mais simples. No parâmetro `FieldInfo`, os arrays de dois elementos consistem de um número de seqüência (que inicia em 1 para o primeiro campo) e de um `xlColumnDataType` a partir da Tabela 20.1. Nesse exemplo, `Array (2, 1)` quer dizer simplesmente 'o segundo campo é de tipo geral'. `Array (3, 3)` quer dizer 'o terceiro campo é uma data no formato M-D-A'. O código é mais longo porque especifica de modo explícito que cada possível delimitador é configurado como `False`. Uma vez que `False` é o padrão para todos os delimitadores, você realmente precisará utilizar um. O código seguinte é equivalente a um delimitador:

```
Workbooks.OpenText Filename:= "C:\vendas.txt", DataType:=xlDelimited, Comma:=True,FieldInfo:=Array(Array(1, 1), _
Array(2, 1), Array(3, 3), Array(4, 1),Array(5, 1), Array(6, 1), Array(7, 1), Array(8, 1))
```

Por fim, para tornar o código mais legível, você pode utilizar os nomes de constante em vez do número de código:

```
Workbooks.OpenText Filename:="C:\vendas.txt", DataType:=xlDelimited, Comma:=True, FieldInfo:=Array(Array(1, _
    xlGeneralFormat), Array(2, xlGeneralFormat), Array(3, xlMDYFormat), Array(4, xlGeneralFormat), Array(5, _
    xlGeneralFormat), Array(6, xlGeneralFormat), Array(7, xlGeneralFormat), Array(8, xlGeneralFormat))
```

O Excel tem opções predefinidas para a leitura de arquivos em que os campos são delimitados por tabulações, ponto-e-vírgula, vírgulas ou espaços. De fato, o Excel pode tratar tudo como um delimitador. Se alguém enviasse texto *delimitado* por barras verticais, você configuraria o parâmetro `Other` como `True` e especificaria um parâmetro `OtherChar`:

```
Workbooks.OpenText Filename:= "C:\vendas.txt", Origin:=437,DataType:=xlDelimited, Other:=True, OtherChar:= "¦", _
    FieldInfo:=...
```

Lendo arquivos de texto com mais de 1.084.576 linhas

Se utilizar o Assistente de Importação de Texto para ler um arquivo com mais de 1.084.576 linhas de dados, você obterá um erro que informa que 'o arquivo não foi carregado completamente'. Isso quer dizer que apenas as primeiras 1.084.576 linhas do arquivo foram carregadas corretamente.

Se utilizar `Workbooks.OpenText` para abrir um arquivo com mais de 1.084.576 linhas de dados, você não receberá nenhuma indicação de que o arquivo não foi completamente carregado. O Excel 2007 carrega as primeiras 1.084.576 linhas e permite que a macro continue a executar. A única indicação real de que há um problema é se alguém notar que os relatórios não contêm informações sobre todas as vendas. Se você achar que seus arquivos nunca ficarão tão grandes assim, seria bom verificar se a célula A1084576 está em branco depois de importar. Se estiver, provavelmente o arquivo inteiro não foi carregado.

Lendo arquivos de texto uma linha por vez

Você poderia encontrar um arquivo de texto com mais de 1.084.576 linhas. Quando isso acontece, a alternativa é ler o arquivo de texto uma linha por vez. O código para fazer isso é o mesmo código que você aprendeu em suas primeiras aulas de BASIC.

Você precisa abrir o arquivo para `INPUT` como `#1`. Pode, então, utilizar a instrução `Line Input #1` para ler uma linha do arquivo em uma variável. O código seguinte abre vendas.txt, lê dez linhas do arquivo nas dez primeiras células da planilha e fecha o arquivo:

```
Sub Import10()
    ThisFile = "C\vendas.txt"
    Open ThisFile For Input As #1
    For i = 1 To 10
        Line Input #1, Data
        Cells(i, 1).Value = Data
    Next i
    Close #1
End Sub
```

Em vez de ler apenas dez registros, você vai querer ler até chegar ao fim do arquivo. Uma variável chamada `EOF` é atualizada automaticamente pelo Excel. Se abrir um arquivo para entrada como `#1`, verificar `EOF(1)` informará se você leu ou não até o último registro.

Utilize um loop `Do...While` para continuar lendo registros até você alcançar o fim do arquivo:

```
Sub ImportAll()
    ThisFile = "C:\vendas.txt"
    Open ThisFile For Input As #1
    Ctr = 0
    Do
        Line Input #1, Data
        Ctr = Ctr + 1
        Cells(Ctr, 1).Value = Data
    Loop While EOF(1) = False
    Close #1
End Sub
```

Depois de ler registros com código como esse, você notará na Figura 20.9 que os dados não são analisados sintaticamente em colunas. Todos os campos estão na Coluna A do arquivo.

A1 fx 1,ABC Cliente,12345,31/03/2005

	A	B	C	D	E	F	G
1	1,ABC Cliente,12345,31/03/2005						
2	2,ABC Cliente,12345,31/03/2005						
3	3,ABC Cliente,12345,31/03/2005						
4	4,ABC Cliente,12345,31/03/2005						
5	5,ABC Cliente,12345,31/03/2005						
6	6,ABC Cliente,12345,31/03/2005						
7	7,ABC Cliente,12345,31/03/2005						
8	8,ABC Cliente,12345,31/03/2005						
9	9,ABC Cliente,12345,31/03/2005						
10	10,ABC Cliente,12345,31/03/2005						
11	11,ABC Cliente,12345,31/03/2005						
12	12,ABC Cliente,12345,31/03/2005						
13	13,ABC Cliente,12345,31/03/2005						
14	14,ABC Cliente,12345,31/03/2005						

Figura 20.9
Quando você estiver lendo um arquivo de texto uma linha por vez, todos os campos de dados terminarão em uma longa entrada na Coluna A.

Utilize o método `TextToColumns` para analisar sintaticamente os registros em colunas. Os parâmetros para `TextToColumns` são quase idênticos ao método `OpenText`:

```
Cells(1, 1).Resize(Ctr, 1).TextToColumns Destination:=Range("A1"), _
DataType:=xlDelimited, Comma:=True, FieldInfo:=Array(Array(1, _
xlGeneralFormat), Array(2, xlMDYFormat), Array(3, xlGeneralFormat), _
Array(4, xlGeneralFormat), Array(5, xlGeneralFormat), Array(6, _
xlGeneralFormat), Array(7,xlGeneralFormat), Array(8, xlGeneralFormat), _
Array(9, xlGeneralFormat), Array(10,xlGeneralFormat), Array(11, _
xlGeneralFormat))
```

ATENÇÃO

Para o restante de sua sessão Excel, o programa se lembrará das configurações de delimitador. Há um bug (recurso?) irritante no Excel. Depois que o programa é informado de que você está utilizando uma vírgula ou uma tabulação como delimitador, toda vez que você tentar colar dados da área de transferência para o Excel, estes serão analisados sintaticamente de modo automático pelos delimitadores especificados no método `OpenText`. Portanto, se você tentou colar algum texto que inclui o cliente ABC, Inc., o texto será analisado sintaticamente, de modo automático, em duas colunas, com texto até ABC em uma coluna e Inc. na próxima coluna.

Em vez de codificar manualmente (*hard-code*) que você está utilizando o designador `#1` para abrir o arquivo de texto, é mais seguro utilizar a função `FreeFile`. Isso retorna um inteiro que representa o próximo número de arquivo disponível para utilização pela instrução `Open`. O código completo para ler um arquivo de texto com menos que 1.084.576 linhas é assim:

```
Sub ImportAll()
    ThisFile = "C:\vendas.txt"
    FileNumber = FreeFile
    Open ThisFile For Input As #FileNumber
    Ctr = 0
    Do
        Line Input #FileNumber, Data
        Ctr = Ctr + 1
        Cells(Ctr, 1).Value = Data
    Loop While EOF(FileNumber) = False
    Close #FileNumber
    Cells(1, 1).Resize(Ctr, 1).TextToColumns Destination:=Range("A1"),DataType:=xlDelimited, Comma:=True, _
        FieldInfo:=Array(Array(1, xlGeneralFormat),Array(2, xlMDYFormat), Array(3, xlGeneralFormat),Array(4, _
        xlGeneralFormat), Array(5, xlGeneralFormat),Array(5, xlGeneralFormat), Array(6, xlGeneralFormat), _
        Array(7, xlGeneralFormat), Array(8, xlGeneralFormat),Array(9, xlGeneralFormat), Array(10, _
        xlGeneralFormat),Array(10, xlGeneralFormat), Array(11, xlGeneralFormat))
End Sub
```

Lendo arquivos de texto com mais de 1.084.576 linhas

Você pode utilizar o método `Line Input` para ler um arquivo de texto grande. Minha estratégia é ler as linhas nas células A1:A1084575 e, então, começar a ler linhas adicionais na célula AA2. Começo na linha 2, no segundo conjunto, para que os títulos possam ser copiados da Linha 1 do primeiro dataset. Se o arquivo for suficientemente grande para preencher a coluna AA, mova para BA2, CA2 e assim por diante. Além disso, paro de escrever colunas quando chego à linha 1084574, deixando duas linhas em branco na parte inferior. Isso assegura que o código `Cells(Rows.Count, 1)""".End(xlup).Row` localize a linha final. O seguinte código lê um arquivo de texto grande em vários conjuntos de colunas:

```
Sub ReadLargeFile()
    ThisFile = "C:\vendas.txt"
    FileNumber = FreeFile
    Open ThisFile For Input As #FileNumber

    NextRow = 1
    NextCol = 1
    Do While Not EOF(1)
        Line Input #FileNumber, Data
        Cells(NextRow, NextCol).Value = Data
        NextRow = NextRow + 1
        If NextRow = (Rows.Count -2) Then
            ' Analisa sintaticamente esses registros
            Range(Cells(1, NextCol), Cells(Rows.Count, NextCol)).TextToColumnsDestination:=Cells(1, NextCol),_
                DataType:=xlDelimited,Comma:=True, FieldInfo:=Array(Array(1, xlGeneralFormat),Array(2, _
                xlMDYFormat), Array(3, xlGeneralFormat),Array(4, xlGeneralFormat), Array(5, _
                xlGeneralFormat),Array(6, xlGeneralFormat), Array(7, xlGeneralFormat),Array(8, _
                xlGeneralFormat), Array(9, xlGeneralFormat),Array(10, xlGeneralFormat), Array(11, _
                xlGeneralFormat))
            ' Copia os títulos de seção 1
            If NextCol > 1 Then
                Range("A1:K1").Copy Destination:=Cells(1, NextCol)
            End If
            ' Configura a próxima seção
            NextCol = NextCol + 26
            NextRow = 2
        End If
    Loop
    Close #FileNumber
```

```
    ' Analisa sintaticamente a seção final de registros
    FinalRow = NextRow - 1
    If FinalRow = 1 Then
        ' Trata se o arquivo tinha coincidentemente 1.084.574 linhas exatas
        NextCol = NextCol - 26
    Else
        Range(Cells(2, NextCol), Cells(FinalRow, NextCol)).TextToColumnsDestination:=Cells(1, NextCol), _
            DataType:=xlDelimited,Comma:=True, FieldInfo:=Array(Array(1, xlGeneralFormat),Array(2, _
            xlMDYFormat), Array(3, xlGeneralFormat),Array(4, xlGeneralFormat), Array(5, xlGeneralFormat), _
            Array(6, xlGeneralFormat), Array(7, xlGeneralFormat),Array(8, xlGeneralFormat), Array(9, _
            xlGeneralFormat),Array(10, xlGeneralFormat), Array(11, xlGeneralFormat))
        If NextCol > 1 Then
            Range("A1:K1").Copy Destination:=Cells(1, NextCol)
        End If
    End If

    DataSets = (NextCol - 1) / 26 + 1

End Sub
```

Em geral, escrevo a variável `DataSets` para uma célula nomeada em algum lugar na pasta de trabalho, para que eu saiba mais tarde quantos datasets há na planilha.

Como você pode imaginar, é possível, utilizando esse método, ler 683.281.620 linhas de dados em uma única planilha. O código que você usou primeiramente para filtrar e informar os dados torna-se agora mais complexo. Talvez você se veja criando tabelas dinâmicas de cada conjunto de coluna para criar um resumo do conjunto de dados e, por fim, resumindo todas as tabelas de resumo em uma tabela dinâmica final.

Em algum ponto, você precisa ver se o aplicativo realmente pertence ao Access ou se os dados devem ser armazenados no Access com um front-end do Excel, como discutido no Capítulo 21, "Utilizando o Access como o back-end para aprimorar o acesso multiusuário aos dados".

Gravando arquivos de texto

O código para gravar arquivos de texto é semelhante ao código para ler arquivos desse tipo. Você precisa abrir um arquivo específico para saída como `#1`. Então, à medida que itera pelos vários registros, você os grava no arquivo utilizando a instrução `Print #1`.

Antes de abrir um arquivo para saída, certifique-se de que todos os exemplos anteriores do arquivo foram excluídos. Você pode utilizar a instrução `Kill` para excluir um arquivo. `Kill` retorna um erro se o arquivo não estava lá em primeiro lugar. Então, você deve utilizar `On Error Resume Next` para evitar um erro.

O código seguinte grava um arquivo de texto para utilização por outro aplicativo:

```
Sub WriteFile()
    ThisFile = "C:\Resultados.txt"

    ' Exclui cópia do arquivo de ontem
    On Error Resume Next
    Kill (ThisFile)
    On Error GoTo 0

    ' Abre o arquivo
    Open ThisFile For Output As #1
    FinalRow = Range("A65536").End(xlUp).Row
    ' Grava o arquivo
    For j = 1 To FinalRow
        Print #1, Cells(j, 1).Value
    Next j
End Sub
```

Esse é um exemplo relativamente trivial. Você pode utilizar esse método para gravar qualquer tipo de arquivo baseado em texto. O código no final do Capítulo 16, "Lendo e gravando na Web", utiliza o mesmo conceito para gravar arquivos HTML.

Próximos passos

Às vezes, você se encontrará gravando arquivos sem necessidade — para importar dados de outro sistema ou produzir dados compatíveis com outro sistema. Utilizar arquivos de texto é um método lento para ler e gravar dados. No Capítulo 21 você aprenderá como gravar para acessar arquivos de banco de dados multidimensional (Multidimensional Database – MDB). Esses arquivos são mais rápidos, indexáveis e permitem acesso multiusuário aos dados.

Utilizando o Access como o back-end para aprimorar o acesso multiusuário aos dados

21

NESTE CAPÍTULO

O exemplo quase no final do Capítulo 20, "Processamento de arquivo de texto", propôs um método para armazenar 683 milhões de registros em uma planilha do Excel. Em algum ponto, você precisa admitir que, apesar de o Excel ser o maior produto do mundo, há uma hora de mudar para o Access e tirar proveito dos arquivos MDB (Multidimensional Database) do Access.

Mesmo antes de você ter mais de um milhão de linhas, outra razão atraente para utilizar arquivos de dados MDB é permitir o acesso multiusuário a dados sem as dores de cabeça associadas com pastas de trabalho compartilhadas.

O Microsoft Excel oferece uma opção para compartilhar uma pasta de trabalho, mas você automaticamente perde diversos recursos importantes do Excel ao fazer isso. Depois de compartilhar uma pasta de trabalho, você não pode utilizar subtotais automáticos, tabelas dinâmicas, modo Agrupar e Realçar, cenários, proteção, AutoFormatação, Estilos, Imagens, Adicionar Gráficos ou Inserir Planilhas.

Se utilizar um front-end Excel VBA e armazenar dados em um banco de dados MDB, você terá o melhor dos dois mundos: o poder e a flexibilidade do Excel e a capacidade de acesso multiusuário disponível no Access.

> **NOTA** O MDB é o formato de arquivo oficial tanto do Microsoft Access como do Microsoft Visual Basic. Isso significa que podemos implantar uma solução do Excel que lê e grava um MDB para clientes que não têm o Microsoft Access. Evidentemente, isso ajuda se você, como desenvolvedor, tiver uma cópia do Access, porque poderá utilizar o front-end desse programa para configurar tabelas e consultas.

ADO Versus DAO

Por vários anos, a Microsoft recomendou o DAO (Data Access Object) para acessar dados em banco de dados externo. O DAO se popularizou muito e uma grande quantidade de código foi escrita para esse tipo de método. Quando a Microsoft liberou o Excel 2000, ela começava a investir no ADO (ActiveX Data Objects). Os conceitos são semelhantes e a sintaxe apresenta apenas uma ligeira diferença. Uso o mais novo ADO neste capítulo. Saiba que, se começasse a percorrer o código escrito um tempo atrás, você se depararia com o código DAO. Exceto por algumas alterações de sintaxe, tanto o código para o ADO como para o DAO são semelhantes. Se descobrir que você tem de depurar algum código antigo utilizando o DAO, consulte os artigos do Microsoft Knowledge Base, localizados no endereço a seguir, que discutem as diferenças entre esses códigos:

```
http://support.microsoft.com/default.aspx?scid=KB;EN-US;q225048&
```

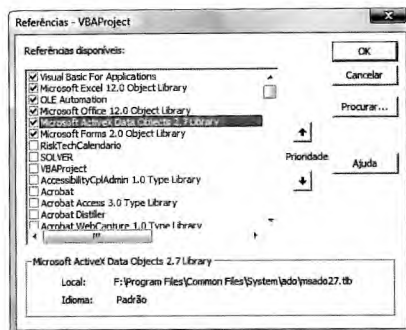
Os dois próximos artigos fornecem a 'Pedra de Rosetta' entre o DAO e o ADO. O código ADO é mostrado no seguinte site Web:

```
http://support.microsoft.com/default.aspx?scid=KB;EN-US;q146607&
```

O código DAO equivalente é mostrado aqui:

```
http://support.microsoft.com/default.aspx?scid=KB;EN-US;q142938&
```

Para utilizar qualquer código neste capítulo, abra o Editor do VB. Selecione Ferramentas, Referências, no menu principal e, então, selecione Microsoft ActiveX Data Objects 2.8 Library (ou versão superior) na lista Referências Disponíveis, como mostrado na Figura 21.1.

Figura 21.1
Para ler ou gravar um arquivo MDB do Access, adicione Microsoft ActiveX Data Objects 2.8 Library (ou versão superior).

Estudo de caso

Linda e Janine são duas compradoras de uma rede varejista de lojas. Toda manhã, elas importam dados das caixas registradoras para obter informações atuais sobre as vendas e o estoque de 2.000 estilos. Durante o dia, qualquer uma das compradoras pode inserir transferências de estoque de uma loja para a outra. Seria ideal se Linda pudesse ver as transferências pendentes inseridas por Janine e vice-versa.

As duas compradoras têm um aplicativo Excel que executa o VBA em sua área de trabalho. Elas importam os dados das caixas registradoras e têm rotinas VBA que facilitam a criação de relatórios de tabela dinâmica para ajudá-las a tomar decisões de compra.

Tentar armazenar os dados de transferência em um arquivo Excel comum causa problemas. Quando alguma delas tentar gravar o arquivo Excel, este se torna o arquivo de leitura para a outra compradora. Com uma pasta de trabalho compartilhada, o Excel desativa a capacidade de criar tabelas dinâmicas, e isso é necessário no aplicativo das compradoras.

Nem Linda nem Janine têm a versão profissional do Office; então, elas não precisam executar o Access na área de trabalho de seus computadores.

A solução é produzir um banco de dados Access em uma unidade de rede que tanto Linda como Janine possam ver:

1. Utilizando o Access em outro PC, produza um novo banco de dados chamado `transfers.mdb` e adicione uma tabela chamada `tblTransfer`, como mostrado na Figura 21.2.

Nome do campo	Tipo de dados
ID	Numeração Automátic
Style	Texto
FromStore	Número
ToStore	Número
Qty	Número
TDate	Data/Hora
Sent	Sim/Não
Receive	Sim/Não

Figura 21.2
Múltiplas pessoas que usam suas próprias pastas de trabalho Excel irão ler essa tabela e gravar nela dentro de um arquivo MDB, em uma unidade de rede.

2. Mova o arquivo Transfers.mdb para uma unidade de rede. Você pode achar que essa pasta comum utiliza diferentes mapeamentos de letra de unidade em cada máquina. Talvez seja H:\Common\ na máquina da Linda e I:\Common\ na máquina de Janine.

3. Em ambos os computadores, vá para o Editor VB e, em Ferramentas, Referências, adicione uma referência a ActiveX Data Objects 2.8 Library (ou versão superior).

4. Em ambos os aplicativos, localize uma célula isolada para armazenar o caminho para transfer.mdb. Nomeie essa célula TPath.

O restante deste capítulo fornece o código necessário para permitir ao aplicativo ler ou gravar dados da tabela tblTransfer.

O aplicativo fornece acesso multiusuário quase transparente para ambas as compradoras. Tanto Linda como Janine podem, ao mesmo tempo, ler a tabela ou gravar dados nela. A única ocasião em que um conflito ocorreria é se ambas tentassem atualizar o mesmo registro ao mesmo tempo.

Exceto pela referência da célula isolada no caminho para transfers.mdb, nenhuma compradora está ciente de que seus dados estão sendo armazenados em uma tabela do Access *compartilhada* e nenhum computador precisa ter o Access instalado.

As ferramentas ADO

Você encontra vários termos ao utilizar o ADO para se conectar a uma origem de dados externa.

- **Recordset** — Ao conectar-se com um banco de dados do Access, o recordset será uma tabela no banco de dados ou uma consulta no banco de dados. A maioria dos métodos ADO referenciará o recordset. Você também pode criar sua própria consulta durante esse processo, isto é, instantaneamente (*on the fly*). Nesse caso, escreva uma instrução SQL para extrair apenas um subconjunto de registros de uma tabela.
- **Conexão** — Define o caminho para o banco de dados e para o tipo de banco de dados. No caso dos bancos de dados Access, especifique que a conexão está utilizando o Microsoft Jet Engine.
- **Cursor** — Pense no cursor como um ponteiro que monitora os registros que estão sendo utilizados no banco de dados. Há vários tipos de cursores e dois locais nos quais eles são localizados (descritos nos próximos itens).
- **Tipos de cursor** — O cursor dinâmico é o mais flexível. Se você definir um recordset e alguém atualizar uma linha na tabela enquanto um cursor dinâmico estiver ativo, este conhecerá o registro atualizado. Embora esse tipo seja o mais flexível, ele requer mais overhead. Se seu banco de dados não tiver uma grande quantidade de transações, você poderá especificar um cursor estático — esse tipo de cursor retorna um instantâneo dos dados na hora em que o cursor é estabelecido.
- **Localização do cursor** — O cursor pode ser localizado no cliente ou no servidor. Para um banco de dados Access residir na unidade de disco, uma localização de servidor para o cursor significa que o Access Jet Engine em seu computador está controlando o cursor. Quando você especificar uma localização de cliente para o cursor, sua sessão Excel estará controlando o cursor. Em um conjunto de dados externo muito grande, seria melhor permitir ao servidor controlar o cursor. Para conjuntos pequenos, um cursor de cliente é mais rápido.
- **Tipo de bloqueio** — O ponto deste capítulo inteiro é permitir que múltiplas pessoas acessem um único conjunto de dados simultaneamente. O tipo de bloqueio define como o ADO evitará quedas quando duas pessoas tentarem atualizar o registro ao mesmo tempo. Com um tipo de bloqueio otimista, um registro individual será bloqueado apenas quando você tentar atualizar o registro. Se o aplicativo estiver fazendo 90 por cento de leituras e só ocasionalmente atualizações, então um bloqueio otimista será perfeito. Mas se souber que toda vez que ler um registro você o atualizará em seguida, então o ideal é um tipo de bloqueio pessimista. Com os bloqueios pessimistas, o registro é bloqueado assim que você o lê. Se você souber que nunca gravará novamente no banco de dados, poderá utilizar um bloqueio de leitura. Isso permite ler os registros sem impedir que os outros gravem neles.

Os objetos primários necessários para acessar dados em um arquivo MDB são uma conexão e um recordset ADO.

A conexão ADO define o caminho para o banco de dados e especifica que a conexão é baseada no Microsoft Jet Engine.

Depois de estabelecer a conexão com o banco de dados, você normalmente utilizará essa conexão para definir um recordset, que pode ser uma tabela ou um subconjunto de registros na tabela ou uma consulta predefinida no banco de dados Access.

Para abrir um recordset, você tem de especificar a conexão e os valores para os parâmetros `CursorType`, `CursorLocation`, `LockType` e `Options`. No caso de serem apenas dois usuários tentando acessar a tabela por vez, geralmente utilizo um cursor dinâmico e um tipo de bloqueio otimista. Para conjuntos de dados grandes, o valor `adUseServer` da propriedade `CursorLocation` permite ao servidor de banco de dados processar registros sem consumir RAM na máquina cliente. Se você tiver um conjunto pequeno, talvez seja mais rápido utilizar `adUseClient` para `CursorLocation`. Quando o recordset estiver aberto, todos os registros serão transferidos para a memória da máquina cliente. Isso permite navegação mais rápida de registro a registro.

Ler dados do banco de dados Access é fácil. Você pode utilizar o método `CopyFromRecordset` para copiar todos os registros selecionados do recordset para uma área em branco da planilha.

Para adicionar um registro à tabela Access, utilize o método `AddNew` ao recordset. Em seguida, especifique o valor para cada campo na tabela e utilize o método `Update` para fazer as alterações no banco de dados.

Para excluir um registro da tabela, você pode usar uma consulta pass-through, que exclui registros que correspondem a certo critério.

> **NOTA** Se alguma vez você se frustrou com o ADO e pensou "se ao menos pudesse abrir o Access, eu criaria uma rápida instrução SQL para fazer exatamente o que preciso", então a consulta *pass-through* foi feita para você. Em vez de usar o ADO para ler todos os registros, a consulta *pass-through* envia uma solicitação para o banco de dados executar a instrução SQL que seu programa constrói. Isso permite tratar efetivamente todas as tarefas que seu banco de dados poderia suportar, mas que não são tratadas pelo ADO. Os tipos de instruções de SQL tratadas pela consulta *pass-through* são dependentes do banco de dados ao qual você está conectado.

Outras ferramentas disponíveis permitem que você confirme se uma tabela existe ou se um campo em particular existe em uma tabela. Você também pode utilizar o VBA para adicionar novos campos para a definição de uma tabela instantaneamente.

Adicionando um registro ao banco de dados

Voltando ao estudo de caso apresentado anteriormente no capítulo, o aplicativo que estamos criando tem um userform no qual as compradoras podem inserir transferências. Para simplificar ao máximo as chamadas para o banco de dados Access, uma série de módulos de utilitário trata a conexão ADO com o banco de dados. Dessa maneira, o código de userform pode simplesmente chamar `AddTransfer(Style, FromStore, ToStore, Qty)`.

A técnica para adicionar registros, depois que a conexão é definida, é assim:

1. Abra um recordset que aponta para a tabela. No código a seguir, veja as seções comentadas `Open the Connection`, `Define the Recordset` e `Open the Table`.
2. Utilize `AddNew` para adicionar um novo registro.
3. Atualize cada campo no novo registro.
4. Utilize `Update` para atualizar o recordset.
5. Feche o recordset e, então, feche a conexão.

O próximo código adiciona um novo registro à tabela tblTransfer:

```
Sub AddTransfer(Style As Variant, FromStore As Variant,ToStore As Variant, Qty As Integer)
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset

    MyConn = "J:\transfers.mdb"

    ' Abre a conexão
    Set cnn = New ADODB.Connection
    With cnn
        .Provider = "Microsoft.Jet.OLEDB.4.0"
        .Open MyConn
    End With

    ' Define o recordset
    Set rst = New ADODB.Recordset
    rst.CursorLocation = adUseServer

    ' Abre a tabela
    rst.Open Source:="tblTransfer",ActiveConnection:=cnn,CursorType:=adOpenDynamic,LockType:=adLockOptimistic,Op
tions:=adCmdTable

    ' Adiciona um registro
    rst.AddNew

    ' Configura os valores para os campos. Os quatro primeiros campos
    ' são passados do userform que chama. O campo de data
    ' é preenchido com a data atual.
    rst("Style") = Style
    rst("FromStore") = FromStore
    rst("ToStore") = ToStore
    rst("Qty") = Qty
    rst("tDate") = Date
    rst("Sent") = False
    rst("Receive") = False

    ' Escreva os valores para esse registro
    rst.Update

    ' Fechar
    rst.Close
    cnn.Close

End Sub
```

Recuperando registros do banco de dados

Ler os registros do banco de dados Access é muito fácil. Como define o recordset, você passa uma string SQL para retornar os registros em que está interessado.

> **DICA** Uma excelente maneira de gerar a SQL é projetar uma consulta no Access que recupera os registros. Ao visualizar essa consulta, escolha Visualização de SQL a partir da lista suspensa Visualização na guia Design de Ferramentas de Consulta da Faixa. O Access mostra a instrução SQL adequada para executar essa consulta. Você pode utilizar essa instrução SQL como modelo para construir a string SQL no código VBA.

Depois que o recordset for definido, utilize o método `CopyFromRecordSet` para copiar todos os registros correspondentes do Access para uma área específica da planilha.

A rotina seguinte consulta a tabela Transfer para localizar todos os registros em que o flag `Sent` ainda não estiver configurado como `True`. Os resultados são colocados em uma planilha em branco. As poucas linhas finais exibem os resultados em um userform, para ilustrar como atualizar um registro na próxima seção:

```
Sub GetUnsentTransfers()
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim WSOrig As Worksheet
    Dim WSTemp As Worksheet
    Dim sSQL as String
    Dim FinalRow as Long

    Set WSOrig = ActiveSheet

    'Constrói uma String SQL para obter todos os campos para transferências não enviadas
    sSQL = "SELECT ID, Style, FromStore, ToStore, Qty, tDate FROM tblTransfer"
    sSQL = sSQL & " WHERE Sent=FALSE"

    ' Caminho para Transfers.mdb
    MyConn = "J:\transfers.mdb"

    Set cnn = New ADODB.Connection
    With cnn
        .Provider = "Microsoft.Jet.OLEDB.4.0"
        .Open MyConn
    End With

    Set rst = New ADODB.Recordset
    rst.CursorLocation = adUseServer
    rst.Open Source:=sSQL, ActiveConnection:=cnn,CursorType:=AdForwardOnly, _
        LockType:=adLockOptimistic,Options:=adCmdText

    ' Cria o relatório em uma nova planilha
    Set WSTemp = Worksheets.Add

    ' Adiciona títulos
    Range("A1:F1").Value = Array("ID", "Estilo", "De", "Para", "Qtde", "Data")

    ' Copia do recordset para linha 2
    Range("A2").CopyFromRecordset rst

    ' Fecha a conexão
    rst.Close
    cnn.Close

    ' Formata o relatório
    FinalRow = Range("A65536").End(xlUp).Row

    ' Se não houver nenhum registro, então pára
    If FinalRow = 1 Then
        Application.DisplayAlerts = False
        WSTemp.Delete
```

```
            Application.DisplayAlerts = True
            WSOrig.Activate
            MsgBox "Não há nenhuma transferência para confirmar"
            Exit Sub
        End If

        ' Formata coluna F como uma data
        Range("F2:F" & FinalRow).NumberFormat = "m/d/y"

        ' Mostra o userform - utilizado na próxima seção
        frmTransConf.Show

        ' Exclui a planilha temporária
        Application.DisplayAlerts = False
        WSTemp.Delete
        Application.DisplayAlerts = True

    End Sub
```

O método `CopyFromRecordSet` copia registros que correspondem à consulta SQL para um intervalo na planilha. Observe que você recebe apenas as linhas de dados. Os títulos não acompanham automaticamente. Você deve utilizar o código para escrever os títulos na Linha 1. A Figura 21.3 mostra os resultados.

Figura 21.3
`Range("A2").CopyFromRecordSet` abre registros correspondentes do banco de dados Access na planilha.

	A	B	C	D	E	F
1	ID	Style	From	To	Qty	Date
2	1935	B11275	340000	340000	8	6/9/0
3	1936	B10133	340000	340000	4	6/9/0
4	1937	B15422	340000	340000	5	6/9/0
5	1938	B10894	340000	340000	9	6/9/0
6	1939	B10049	340000	340000	3	6/9/0
7	1941	B18722	340000	340000	10	6/9/0
8	1944	B12886	340000	340000	10	6/9/0
9	1947	B17947	340000	340000	7	6/9/0
10	1950	B16431	340000	340000	9	6/9/0
11	1953	B19857	340000	340000	7	6/9/0
12	1954	B11562	340000	340000	1	6/9/0
13	1955	B19413	340000	340000	2	6/9/0
14	1957	B17370	340000	340000	1	6/9/0
15	1958	B14304	340000	340000	5	6/9/0
16	1959	B19881	340000	340000	5	6/9/0
17	1960	B13722	340000	340000	1	6/9/0
18	1961	B16873	340000	340000	6	6/9/0
19	1962	B14620	340000	340000	7	6/9/0
20	1963	B12306	340000	340000	1	6/9/0
21	1964	B18110	340000	340000	9	6/9/0
22	1965	B15963	340000	340000	4	6/9/0
23	1966	B12256	340000	340000	6	6/9/0
24	1967	B15878	340000	340000	10	6/9/0

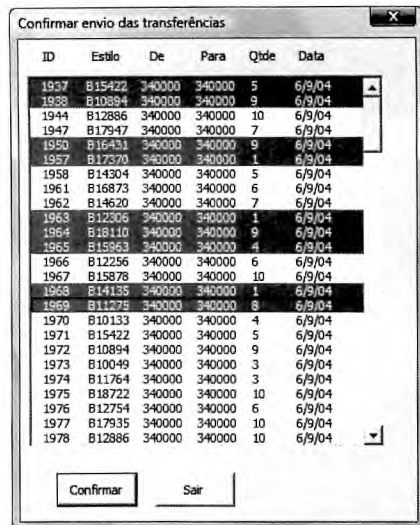
Atualizando um registro existente

Para atualizar um registro existente, você precisa construir um recordset com exatamente um registro. Isso requer que o usuário selecione algum tipo de chave única ao identificar os registros. Depois de abrir o recordset, utilize a propriedade `Fields` para alterar o campo em questão e, então, o método Update para fazer as alterações no banco de dados.

O exemplo anterior retornou um recordset para uma planilha em branco e chamou um userform de frmTransConf. Este formulário utiliza um simples `Userform_Initialize` para exibir o intervalo em uma caixa de listagem grande. As propriedades dessa caixa de listagem têm a propriedade `MultiSelect` configurada como `True`:

```
Private Sub UserForm_Initialize()

    ' Determina quantos registros temos
    FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
    If FinalRow > 1 Then
        Me.lbXlt.RowSource = "A2:F" & FinalRow
    End If

End Sub
```

Depois que o procedimento de inicialização é executado, os registros não confirmados são exibidos em uma caixa de listagem. O planejador logístico é capaz de marcar todos os registros que foram realmente enviados, como mostrado na Figura 21.4.

Figura 21.4
Esse userform exibe registros específicos do recordset Access. Quando um comprador seleciona certos registros e, então, escolhe o botão Confirmar, você tem de utilizar o método `Update` do ADO para atualizar o campo Sent nos registros selecionados.

A seguir, temos o código anexado ao botão Confirmar. Incluir o campo ID nos campos retornados no exemplo anterior é importante quando se quer restringir as informações a um único registro:

```
Private Sub cbConfirm_Click()
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset

    ' Se nada estiver selecionado, adverte-os
    CountSelect = 0
    For x = 0 To Me.lbXlt.ListCount - 1
        If Me.lbXlt.Selected(x) Then
            CountSelect = CountSelect + 1
        End If
    Next x

    If CountSelect = 0 Then
        MsgBox "Não há nenhuma transferência selecionada. " & _
            "Para sair sem confirmar nenhuma transferência, utilize Cancelar."
        Exit Sub
    End If

    ' Estabelece uma conexão transfers.mdb
    ' O caminho para Transfer.mdb está no menu
    MyConn = "J:\transfers.mdb"

    Set cnn = New ADODB.Connection

    With cnn
        .Provider = "Microsoft.Jet.OLEDB.4.0"
        .Open MyConn
    End With

    ' Marca como completo
    For x = 0 To Me.lbXlt.ListCount - 1
        If Me.lbXlt.Selected(x) Then
            ThisID = Cells(2 + x, 1).Value
            ' Marca ThisID como completo
            'Constrói string SQL
            sSQL = "SELECT * FROM tblTransfer Where ID=" & ThisID
            Set rst = New ADODB.Recordset
            With rst
                .Open Source:=sSQL, ActiveConnection:=cnn,CursorType:=adOpenKeyset, LockType:=adLockOptimistic
                ' Atualiza o campo
                .Fields("Sent").Value = True
                .Update
                .Close
            End With
        End If
    Next x
```

```
    ' Fecha a conexão
    cnn.Close
    Set rst = Nothing
    Set cnn = Nothing

    ' Fecha o userform
    Unload Me

End Sub
```

Excluindo registros por meio do ADO

Do mesmo modo que atualizar um registro, a chave para excluir registro(s) pode agora escrever um fragmento de SQL para identificar exclusivamente o(s) registro(s) a ser(em) excluído(s). O código seguinte utiliza o método `Execute` para passar o comando `Delete` pelo Access:

```
Public Sub ADOWipeOutAttribute(RecID)
    ' Estabelece uma conexão transfers.mdb
    MyConn = "J:\transfers.mdb"

    With New ADODB.Connection
        .Provider = "Microsoft.Jet.OLEDB.4.0"
        .Open MyConn
        .Execute "Delete From tblTransfer Where ID = " & RecID
        .Close
    End With
End Sub
```

Resumindo registros por meio do ADO

Um dos poderes do Access é administrar consultas de resumo que agrupam um campo específico. Se você construir uma consulta de resumo no Access e examinar a visualização SQL, verá que as consultas complexas podem ser escritas. Uma instrução SQL semelhante pode ser criada no Excel VBA e passada para o Access por meio do ADO.

O seguinte código utiliza uma consulta relativamente complexa para obter um total líquido por loja:

```
Sub NetTransfers(Style As Variant)
    ' Constrói uma tabela de transferências líquidas abertas
    ' nos Estilos AI1
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset

    ' Constrói a consulta SQL maior
    ' Lógica básica: obtém todas as novas transferências recebidas abertas por loja,
    ' união com -1* transferência enviada por loja
    ' Soma essa união por loja e também fornece a data em min.
    ' Uma única chamada a essa macro substituirá 60 chamadas a GetTransferIn, GetTransferOut, TransferAge
    sSQL = "Select Store, Sum(Quantity), Min(mDate) From(SELECT ToStore AS Store, Sum(Qty) AS Quantity,_
        Min(TDate) AS mDate FROM tblTransfer where Style='" & Style& "' AND Receive=FALSE GROUP BY ToStore "_
        sSQL = sSQL & " Union All SELECT FromStore AS Store,Sum(-1*Qty) AS Quantity, Min(TDate) AS mDateFROM _
        tblTransfer where Style='" & Style & "' ANDSent=FALSE GROUP BY FromStore)"
    sSQL = sSQL & " Group by Store"

    MyConn = "J:\transfers.mdb"

    ' Abre a conexão.
    Set cnn = New ADODB.Connection
    With cnn
        .Provider = "Microsoft.Jet.OLEDB.4.0"
        .Open MyConn
    End With

    Set rst = New ADODB.Recordset

    rst.CursorLocation = adUseServer

    ' Abre a primeira consulta
```

```
    rst.Open Source:=sSQL,ActiveConnection:=cnn,CursorType:=AdForwardOnly,LockType:=adLockOptimistic,_
        Options:=adCmdText

    Range("A1:C1").Value = Array("Store", "Qty", "Date")
    ' Retorna resultados da consulta
    Range("A2").CopyFromRecordset rst
    rst.Close
    cnn.Close

End Sub
```

Outros utilitários por meio do ADO

Considere o aplicativo que criamos para nosso estudo de caso; as compradoras agora têm um banco de dados Access localizado na rede, mas, possivelmente, não dispõem de cópia do Access. Seria ideal se você pudesse apresentar alterações no banco de dados Access instantaneamente, à medida que o aplicativo deles abre.

DICA

Se você quer saber como persuadir a pessoa que utiliza o aplicativo a executar essas consultas, considere o uso de uma macro Update oculta na rotina `Workbook_Open` do aplicativo cliente. Tal rotina poderia primeiro confirmar que um campo não existe e, então, adicioná-lo. Para detalhes sobre a mecânica do ocultamento da consulta de atualização na rotina `Workbook_Open`, veja o estudo de caso, "Utilizando uma pasta de trabalho de código oculta para armazenar todas as macros e formulários", no Capítulo 27, "Criando suplementos".

Verificando a existência de tabelas

Se o aplicativo precisar de uma nova tabela no banco de dados, você pode utilizar o código apresentado na próxima seção. Porém, como temos um aplicativo multiusuário, apenas a primeira pessoa que abre esse aplicativo tem de adicionar a tabela instantaneamente. Quando a próxima compradora aparecer, a tabela já pode ter sido adicionada pelo aplicativo da primeira compradora.

Esse código utiliza o método `OpenSchema` para, de fato, consultar o esquema de banco de dados:

```
Function TableExists(WhichTable)
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim fld As ADODB.Field
    TableExists = False


    ' O caminho para Transfer.mdb está no menu
    MyConn = "J:\transfers.mdb"

    Set cnn = New ADODB.Connection

    With cnn
        .Provider = "Microsoft.Jet.OLEDB.4.0"
        .Open MyConn
    End With

    Set rst = cnn.OpenSchema(adSchemaTables)

    Do Until rst.EOF
        If LCase(rst!Table_Name) = LCase(WhichTable) Then
            TableExists = True
            GoTo ExitMe
        End If
        rst.MoveNext
    Loop

ExitMe:
    rst.Close
    Set rst = Nothing
    ' Fecha a conexão
    cnn.Close

End Function
```

Verificando a existência de um campo

Às vezes, você pode querer adicionar um novo campo a uma tabela existente. Novamente, esse código utiliza o método `OpenSchema`, mas dessa vez vê as colunas nas tabelas:

```
Function ColumnExists(WhichColumn, WhichTable)
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim WSOrig As Worksheet
    Dim WSTemp As Worksheet
    Dim fld As ADODB.Field
    ColumnExists = False

    ' Caminho para Transfers.mdb está no menu
    MyConn = ActiveWorkbook.Worksheets("Menu").Range("TPath").Value
    If Right(MyConn, 1) = "\" Then
        MyConn = MyConn & "transfers.mdb"
    Else
        MyConn = MyConn & "\transfers.mdb"
    End If

    Set cnn = New ADODB.Connection

    With cnn
        .Provider = "Microsoft.Jet.OLEDB.4.0"
        .Open MyConn
    End With

    Set rst = cnn.OpenSchema(adSchemaColumns)

    Do Until rst.EOF
        If LCase(rst!Column_Name) = LCase(WhichColumn) AndLCase(rst!Table_Name) = LCase(WhichTable) Then
            ColumnExists = True
            GoTo ExitMe
        End If
        rst.MoveNext
    Loop

ExitMe:
    rst.Close
    Set rst = Nothing
    ' Fecha a conexão
    cnn.Close

End Function
```

Adicionando uma tabela instantaneamente

Esse código utiliza uma consulta pass-through para dizer ao Access que execute um comando `Create Table`:

```
Sub ADOCreateReplenish()
    ' Cria tblReplenish
    ' Existem cinco campos:
    ' Estilo
    ' A = Reabastecimento automático para A
    ' B = Nível de reabastecimento automático para as lojas B
    ' C = Nível de reabastecimento automático para as lojas C
    ' RecActive = campo Sim/Não
    Dim cnn As ADODB.Connection
    Dim cmd As ADODB.Command

    ' Define a conexão
    MyConn = "J:\transfers.mdb"

    ' Abre a conexão
    Set cnn = New ADODB.Connection
    With cnn
        .Provider = "Microsoft.Jet.OLEDB.4.0"
        .Open MyConn
    End With
```

```
    Set cmd = New ADODB.Command
    Set cmd.ActiveConnection = cnn
    'Cria a tabela
    cmd.CommandText = "CREATE TABLE tblReplenish (Style Char(10) Primary Key,A int, B  int, C Int, RecActive
YesNo)"
    cmd.Execute , , adCmdText
    Set cmd = Nothing
    Set cnn = Nothing
    Exit Sub
End Sub
```

Adicionando um campo instantaneamente

Se determinar que um campo não existe, você pode utilizar uma consulta pass-through, adicionando um campo à tabela:

```
Sub ADOAddField()
    ' Adiciona um campo e grp a tblReplenish
    Dim cnn As ADODB.Connection
    Dim cmd As ADODB.Command

    ' Define a conexão
    MyConn = "J:\transfers.mdb"
    End If

    ' Abre a conexão
    Set cnn = New ADODB.Connection
    With cnn
        .Provider = "Microsoft.Jet.OLEDB.4.0"
        .Open MyConn
    End With


   Set cmd = New ADODB.Command
   Set cmd.ActiveConnection = cnn
   'Cria a tabela
   cmd.CommandText = "ALTER TABLE tblReplenish Add Column Grp Char(25)"
   cmd.Execute , , adCmdText
   Set cmd = Nothing
   Set cnn = Nothing

End Sub
```

Próximos passos

No Capítulo 22, “Criando classes, registros e coleções”, você aprenderá a poderosa técnica de configurar seu próprio módulo de classe. Com essa técnica, você poderá configurar seu próprio objeto com seus próprios métodos e propriedades.

Criando classes, registros e coleções

22

NESTE CAPÍTULO

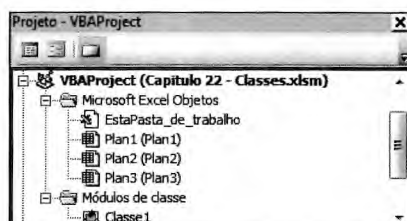
O Excel já dispõe de muitos objetos, mas há ocasiões em que um objeto personalizado seria melhor que o trabalho à mão. Você pode criar objetos personalizados que são usados da mesma maneira que os objetos predefinidos do Excel. Esses objetos especiais são criados em *módulos de classe.*

Os módulos de classe são utilizados para criar objetos personalizados com propriedades e métodos personalizados. Eles podem interceptar eventos de aplicativo, de gráfico incorporado, de controle ActiveX, entre outros.

Inserindo um módulo de classe

No Editor do VB, selecione Inserir, Módulo de Classe. Um novo módulo, Classe1, é adicionado à pasta de trabalho VBAProject e pode ser visto na janela Project Explorer (veja Figura 22.1). Duas coisas devem ser manitdas em mente em relação aos módulos de classe:

- Todo objeto personalizado deve ter seu próprio módulo (a interceptação de evento pode compartilhar um módulo).
- O módulo de classe deve ser renomeado para refletir o objeto personalizado.

Figura 22.1
Os objetos personalizados são criados em módulos de classe.

Interceptando eventos de aplicativo e gráficos incorporados

O Capítulo 9, "Programação de eventos", mostrou como certas ações em pastas de trabalho, planilhas e gráficos não incorporados poderiam ser interceptadas e utilizadas para ativar o código. Revisamos brevemente como configurar um módulo de classe para interceptar eventos de aplicativo e de gráfico. A próxima seção entra em mais detalhes sobre o que foi mostrado naquele capítulo.

Eventos de aplicativo

O evento `Workbook_BeforePrint` é desencadeado quando a pasta de trabalho em que ele reside é impressa. Se quiser executar o mesmo código em todas as pastas de trabalho disponíveis, você tem de copiá-lo em cada pasta de trabalho. De outro modo, poderia utilizar um evento de aplicativo, `Workbook_BeforePrint`, que é desencadeado quando qualquer pasta de trabalho é impressa.

Os eventos de aplicativo já existem, mas um módulo de classe deve ser configurado primeiro, para que ele possa ser visto. Para criar um módulo de classe, siga estes passos:

1. Insira um módulo de classe no projeto. Mude o nome do módulo para algo que faça sentido para você, como `clsAppEvents`. Escolha Visualização, janela Propriedades, para renomear um módulo.

2. Insira o seguinte no módulo de classe:

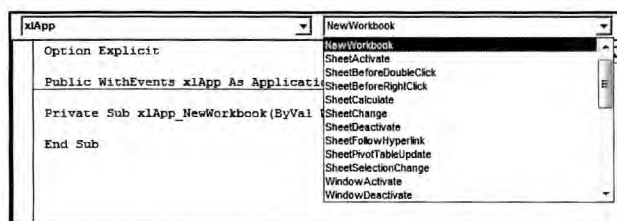
   ```
   Public WithEvents xlApp As Application
   ```

   O nome da variável, `xlApp`, pode ser qualquer nome variável. A palavra-chave `WithEvents` expõe os eventos associados com o objeto Aplicativo.

3. `xlApp` está agora disponível na lista suspensa Objeto desse módulo de classe. Selecione-a na lista suspensa e, então, clique no menu suspenso Procedimento, à direita, para visualizar a lista de eventos disponível para o tipo de objeto (Aplicativo) da variável `xlApp`, como mostrado na Figura 22.2.

➔ Para uma revisão dos vários Eventos de Aplicativo, **consulte** "Eventos de nível de aplicativo", **p. 171**, no Capítulo 9, "Programação de eventos".

Qualquer um dos eventos listados pode ser capturado, exatamente como os eventos de pasta de trabalho e de planilha foram capturados em um capítulo anterior. O exemplo a seguir utiliza o evento `NewWorkbook` para configurar as informações de rodapé automaticamente. Esse código é colocado no módulo de classe, abaixo da linha de declaração `xlApp` que você acabou de adicionar:

Figura 22.2
Os eventos são disponibilizados depois que o objeto é criado.

```
Private Sub xlApp_NewWorkbook(ByVal Wb As Workbook)
Dim wks As Worksheet
With Wb
    For Each wks In .Worksheets
        wks.PageSetup.LeftFooter = "Criado por: " & .Application.UserName
        wks.PageSetup.RightFooter = Now
    Next wks
End With

End Sub
```

O procedimento colocado em um módulo de classe não executa automaticamente, como os eventos na pasta de trabalho ou módulos de planilha executariam. Uma instância do módulo de classe deve ser criada e o objeto Aplicativo, atribuído à propriedade `xlApp`. Depois que isso tiver sido completado, o procedimento `TrapAppEvent` precisa executar. Contanto que `TrapAppEvent` esteja em execução, o rodapé será criado em cada planilha toda vez que uma nova pasta de trabalho for adicionada. Coloque o seguinte em um módulo-padrão:

```
Public myAppEvent As New clsAppEvents

Sub TrapAppEvent()

Set myAppEvent.xlApp = Application

End Sub
```

ATENÇÃO

A interceptação do evento de aplicativo pode ser terminada por qualquer ação que redefina o nível do módulo ou as variáveis públicas. Isso inclui editar código no Editor do VB. Para reiniciar, execute o procedimento que cria o objeto (`TrapAppEvent`).

Nesse exemplo, a declaração `myAppEvent` pública foi colocada em um módulo-padrão com o procedimento `TrapAppEvent`. Para automatizar a execução da interceptação do evento inteiro, todos os módulos poderiam ser transferidos para Personal.xlsb e o procedimento, transferido para um evento `Workbook_Open`. Em qualquer caso, a declaração `Public` de `myAppEvent` deve permanecer em um módulo-padrão para que possa ser compartilhada entre os módulos.

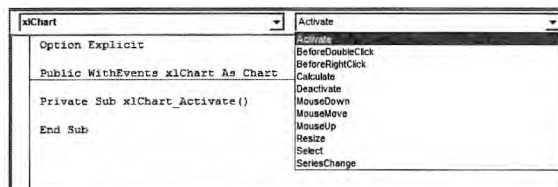
Eventos de gráficos incorporados

Preparar-se para interceptar eventos de gráficos incorporados é o mesmo que se preparar para interceptar eventos de aplicativo. Crie um módulo de classe, insira a declaração pública em um tipo de gráfico, crie um procedimento profissional para o evento desejado e, então, adicione um procedimento de módulo-padrão para iniciar a interceptação. O mesmo módulo de classe utilizado para o evento de aplicativo pode ser utilizado para o evento de gráfico incorporado.

Coloque a linha seguinte na seção de declaração do módulo de classe. Os eventos de gráfico disponíveis são agora visualizáveis (veja Figura 22.3):

```
Public WithEvents xlChart As Chart
```

➔ Para uma revisão dos vários eventos de gráfico, **consulte** "Eventos de planilha de gráfico", **p. 167**, no Capítulo 9, "Programação de eventos".

Figura 22.3 Os eventos de gráfico são disponibilizados depois que a variável de tipo de gráfico foi declarada.

Três eventos são configurados. O evento primário, `MouseDown`, muda a escala do gráfico quando clicamos com o botão direito do mouse ou damos um clique duplo. Como essas ações têm outras ações associadas a elas, são necessários mais dois eventos, `BeforeRightClick` e `BeforeDoubleClick`, para impedir que a ação normal aconteça:

O evento `BeforeDoubleClick` impede que ocorra o resultado normal de um clique duplo:

```
Private Sub xlChart_BeforeDoubleClick(ByVal ElementID As Long,ByVal Arg1 As Long, ByVal Arg2 As Long, Cancel As
Boolean)
Cancel = True
End Sub
```

O evento `BeforeRightClick` impede que ocorra o resultado normal de um clique com o botão direito do mouse:

```
Private Sub xlChart_BeforeRightClick(Cancel As Boolean)
Cancel = True
End Sub
```

Agora que as ações normais do clique duplo e do clique com o botão direito do mouse estão sob controle, `ChartMouseDown` reescreve quais ações devem ser iniciadas por um clique com o botão direito do mouse e com um clique duplo:

```
Private Sub xlChart_MouseDown(ByVal Button As Long, ByVal Shift As Long,ByVal x As Long, ByVal y As Long)
    If Button = 1 Then
       xlChart.Axes(xlValue).MaximumScale =xlChart.Axes(xlValue).MaximumScale - 50
    End If

    If Button = 2 Then
       xlChart.Axes(xlValue).MaximumScale =xlChart.Axes(xlValue).MaximumScale + 50
    End If

End Sub
```

Depois que os eventos tiverem sido configurados no módulo de classe, tudo que resta a fazer é declarar a variável em um módulo-padrão, como a seguir:

```
Public myChartEvent As New clsEvents
```

Então, crie um procedimento que capturará os eventos no gráfico incorporado:

```
Sub TrapChartEvent()

Set myChartEvent.xlChart = Worksheets("EmbedChart").ChartObjects("Chart 2").Chart

End Sub
```

> **NOTA** Os eventos `BeforeDoubleClick` e `BeforeRightClick` só são desencadeados quando o usuário clica na própria área de plotagem. A área em torno da área de plotagem não desencadeia os eventos. Mas o evento `MouseDown` é desencadeado em qualquer lugar no gráfico.

Criando um objeto personalizado

Os módulos de classe são úteis para interceptar eventos, mas também são valiosos porque podem ser utilizados para criar objetos personalizados. Quando você estiver criando um objeto personalizado, o módulo de classe torna-se um modelo das propriedades e métodos do objeto. Para entender isso melhor, vamos criar um objeto empregado para monitorar o nome, o ID, o índice de salário por hora e as horas trabalhadas de um funcionário.

Insira um módulo de classe e renomeie-o como **clsEmployee**. O objeto clsEmployee tem quatro propriedades:

- **EmpName** — Nome do funcionário
- **EmpID** — Código do funcionário
- **EmpRate** — Índice de salário por hora
- **EmpWeeklyHrs** — Horas trabalhadas

As propriedades são variáveis que podem ser declaradas `Private` ou `Public`. Essas propriedades precisam ser acessíveis ao módulo-padrão para que sejam declaradas `Public`. Coloque as seguintes linhas na parte superior do módulo de classe:

```
Public EmpName As String
Public EmpID As String
Public EmpRate As Double
Public EmpWeeklyHrs As Double
```

Os métodos são ações que o objeto pode praticar. No módulo de classe, essas ações assumem a forma de procedimentos e funções. O código a seguir cria um método, `EmpWeeklyPay()`, para o objeto que calcula o pagamento semanal:

```
Public Function EmpWeeklyPay() As Double
EmpWeeklyPay = EmpRate * EmpWeeklyHrs
End Function
```

O objeto está agora completo. Ele tem quatro propriedades e um método. O próximo passo é utilizá-lo em um programa real.

Utilizando um objeto personalizado

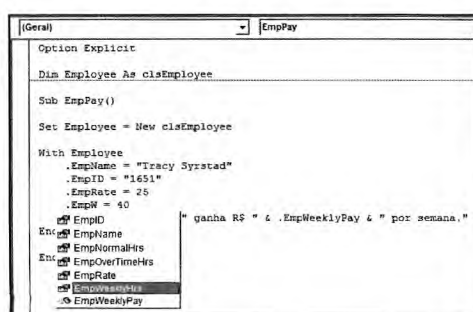
Depois que um objeto personalizado é adequadamente configurado em um módulo de classe, ele pode ser referenciado de outro módulo. Declare uma variável como o tipo de objeto personalizado na seção de declarações:

```
Dim Employee As clsEmployee
```

No procedimento, configure a variável como um objeto New:

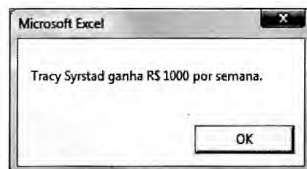
```
Set Employee = New clsEmployee
```

Continue inserindo o restante do procedimento. Enquanto você referencia as propriedades e o método do objeto personalizado, uma dica de tela é exibida, assim como com objetos-padrão do Excel (veja Figura 22.4).

Figura 22.4
As propriedades e o método do objeto personalizado são igualmente acessíveis, como ocorre com objetos-padrão.

```
Option Explicit

Dim Employee As clsEmployee

Sub EmpPay()

Set Employee = New clsEmployee

With Employee
```

```
    .EmpName = "Tracy Syrstad"
    .EmpID = "1651"
    .EmpRate = 25
    .EmpWeeklyHrs = 40
    MsgBox .EmpName & " ganha $" & .EmpWeeklyPay & " por semana."
End With

End Sub
```

O subprocedimento declara um objeto Employee como uma nova instância de clsEmployee. Em seguida, atribui valores às quatro propriedades do objeto e gera uma caixa de mensagem, que exibe o nome do funcionário e o pagamento semanal (veja Figura 22.5). O método do objeto, `EmpWeeklyPay`, é utilizado para gerar o salário exibido.

Figura 22.5
Crie objetos personalizados para tornar o código mais eficiente.

Utilizando `Property Let` e `Property Get` para controlar como os usuários utilizam os objetos personalizados

As variáveis públicas, como declaradas no exemplo anterior, têm propriedades de leitura e/ou gravação. Quando utilizados em um programa, os valores das variáveis podem ser recuperados ou alterados. Para atribuir limitações de leitura/gravação, utilize os procedimentos `Property Let` e `Property Get`.

Os procedimentos `Property Let` nos dão controle de como é possível atribuir valores às propriedades. Os procedimentos `Property Get` controlam como as propriedades são acessadas. No exemplo de objeto personalizado, há uma variável pública para horas semanais. Essa variável é utilizada em um método para calcular o salário semanal, mas não leva em conta o pagamento de horas extras. As variáveis para horas normais e horas extras são necessárias, mas as variáveis devem ser somente-leitura.

Para realizar isso, o módulo de classe deve ser reconstruído. Ele precisa de duas novas propriedades, `EmpNormalHrs` e `EmpOverTimeHrs`. Mas como essas duas propriedades devem ser propriedades somente-leitura, elas não são declaradas como variáveis. Os procedimentos `Property Get` são utilizados para criá-las.

Se `EmpNormalHrs` e `EmpOverTimeHrs` forem propriedades somente-leitura, de qualquer modo, os valores devem ser atribuídos a elas. Seus valores são um cálculo de `EmpWeeklyHrs`. Como `EmpWeeklyHrs` será utilizada para configurar os valores de propriedade dessas duas propriedades de objeto, ela não pode mais ser uma variável pública. Há duas variáveis privadas, `NormalHrs` e `OverHrs`, que são utilizadas dentro dos limites do módulo de classe:

```
Public EmpName As String
Public EmpID As String
Public EmpRate As Double

Private NormalHrs As Double
Private OverHrs As Double
```

Um procedimento Property Let é criado por EmpWeeklyHrs para dividir as horas em horas normais e horas extras:

```
Property Let EmpWeeklyHrs(Hrs As Double)

NormalHrs = WorksheetFunction.Min(40, Hrs)
OverHrs = WorksheetFunction.Max(0, Hrs - 40)

End Property
```

`Property Get EmpWeeklyHrs` soma essas horas e retorna um valor para essa propriedade. Sem ela, não seria possível recuperar um valor a partir de EmpWeeklyHrs:

```
Property Get EmpWeeklyHrs() As Double

EmpWeeklyHrs = NormalHrs + OverHrs

End Property
```

Os procedimentos `Property Get` são criados para que `EmpNormalHrs` e `EmpOverTimeHrs` configurem seus valores. Se você utilizar apenas procedimentos `Property Get`, os valores dessas duas propriedades serão somente-leitura. Só podem receber valores pela propriedade *EmpWeeklyHrs*:

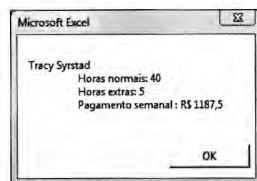
```
Property Get EmpNormalHrs() As Double

EmpNormalHrs = NormalHrs

End Property
Property Get EmpOverTimeHrs() As Double

EmpOverTimeHrs = OverHrs

End Property
```

Por fim, o método `EmpWeeklyPay` é atualizado para refletir as alterações nas propriedades e no objetivo:

```
Public Function EmpWeeklyPay() As Double

EmpWeeklyPay = (EmpNormalHrs * EmpRate) + (EmpOverTimeHrs * EmpRate * 1.5)

End Function
```

Atualize o procedimento no módulo-padrão para tirar proveito das alterações no módulo de classe. A Figura 22.6 mostra a nova caixa de mensagem resultante desse procedimento atualizado:

```
Sub EmpPayOverTime()
Dim Employee As New clsEmployee

With Employee
    .EmpName = "Tracy Syrstad"
    .EmpID = "1651"
    .EmpRate = 25
    .EmpWeeklyHrs = 45
    MsgBox .EmpName & Chr(10) & Chr(9) &"Horas normais: " & .EmpNormalHrs & Chr(10) & Chr(9) &"Horas extras: " &
.EmpOverTimeHrs & Chr(10) & Chr(9) &"Pagamento semanal : $" & .EmpWeeklyPay
End With

End Sub
```

Figura 22.6
Utilize os procedimentos `Property Let` e `Property Get` para obter mais controle sobre as propriedades do objeto personalizado.

Coleções

Até agora, você conseguiu fazer somente um registro por vez do objeto personalizado. Para criar mais, uma coleção é necessária. Uma coleção permite que exista mais de um único registro por vez. Por exemplo, Planilha é membro da coleção Planilhas. Você pode adicionar, remover, contar e referenciar cada planilha em uma pasta de trabalho por item. Essa funcionalidade também está disponível para seu objeto personalizado.

Criando uma coleção em um módulo-padrão

A maneira mais rápida de criar uma coleção é por meio do método `Collection` predefinido. Se configurar uma coleção em um módulo-padrão, você poderá acessar os quatro métodos de coleção-padrão: `Add`, `Remove`, `Count` e `Item`.

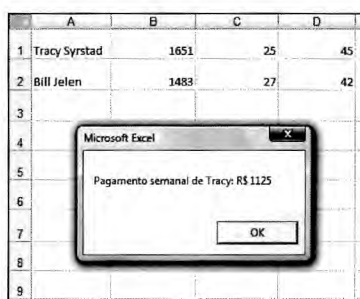
O exemplo seguinte lê uma lista de funcionários fora de uma planilha e em um array. Então processa o array, fornecendo cada propriedade do objeto com um valor e colocando cada registro na coleção:

```
Sub EmpPayCollection()
Dim colEmployees As New Collection
Dim recEmployee As New clsEmployee
Dim LastRow As Integer, myCount As Integer
Dim EmpArray As Variant
```

```
LastRow = ActiveSheet. Cells(ActiveSheet.Rows.Count, 1).End(xlUp).Row
EmpArray = ActiveSheet.Range(Cells(1, 1), Cells(LastRow, 4))

For myCount = 1 To UBound(EmpArray)
    Set recEmployee = New clsEmployee
    With recEmployee
        .EmpName = EmpArray(myCount, 1)
        .EmpID = EmpArray(myCount, 2)
        .EmpRate = EmpArray(myCount, 3)
        .EmpWeeklyHrs = EmpArray(myCount, 4)
        colEmployees.Add recEmployee, .EmpID
    End With
Next myCount

MsgBox "Número de funcionários: " & colEmployees.Count & Chr(10) & _
    "Nome do funcionário(2): " & colEmployees(2).EmpName
MsgBox "Pagamento semanal de Tracy: $" & colEmployees("1651").EmpWeeklyPay

Set recEmployee = Nothing

End Sub
```

A coleção `colEmployees` é declarada como uma nova coleção e o registro `recEmployee`, como uma nova variável do tipo de objeto personalizado.

Depois que as propriedades do objeto recebem valores, o registro `recEmployee` é adicionado à coleção. O segundo parâmetro do método `Add` aplica uma chave única ao registro — nesse caso, o número ID do empregado. Isso permite acessar rapidamente um registro específico, como mostrado pela segunda caixa de mensagem (`colEmployees("1651").EmpWeeklyPay`). Veja Figura 22.7.

Figura 22.7
Registros individuais em uma coleção podem ser facilmente acessados.

> **NOTA** A chave única é um parâmetro opcional. Uma mensagem de erro aparece se uma chave duplicada for inserida.

Criando uma coleção em um módulo de classe

As coleções podem ser criadas em um módulo de classe mas, nesse caso, os métodos inatos da coleção (`Add`, `Remove`, `Count`, `Item`) não estão disponíveis; elas têm de ser criadas no módulo de classe. As vantagens de criar uma coleção em um módulo de classe são que o código inteiro está em um módulo, você tem mais controle sobre o que é feito com a coleção e pode evitar o acesso à coleção.

Insira um novo módulo de classe na coleção e renomeie-a como `clsEmployees`. Declare uma coleção privada para ser utilizada dentro do módulo de classe:

```
Option Explicit
Private AllEmployees As New Collection
```

Adicione as novas propriedades e métodos necessários para fazer a coleção funcionar. Os métodos inatos da coleção estão disponíveis dentro do módulo de classe e podem ser utilizados para criar as propriedades e métodos personalizados:

Insira um método `Add` para adicionar novos itens à coleção:

```
Public Sub Add(recEmployee As clsEmployee)

AllEmployees.Add recEmployee, recEmployee.EmpID

End Sub
```

Insira uma propriedade `Count` para retornar o número de itens na coleção:

```
Public Property Get Count() As Long

Count = AllEmployees.Count

End Property
```

Insira uma propriedade `Items` para retornar a coleção inteira:

```
Public Property Get Items() As Collection

Set Items = AllEmployees

End Property
```

Insira uma propriedade Item para retornar um determinado item da coleção:

```
Public Property Get Item(myItem As Variant) As clsEmployee

Set Item = AllEmployees(myItem)

End Property
```

Insira uma propriedade Remove para remover um item específico da coleção:

```
Public Sub Remove(myItem As Variant)

AllEmployees.Remove (myItem)

End Sub
```

`Property Get` é utilizada com `Count`, `Item` e `Items` porque essas são propriedades somente-leitura. `Item` retorna uma referência a um único membro da coleção, ao passo que `Items` retorna a coleção inteira (para que possa ser utilizado em loops `For Each Next`).

Depois que a coleção for configurada no módulo de classe, um procedimento pode ser escrito em um módulo-padrão para utilizá-lo:

```
Sub EmpAddCollection()
Dim colEmployees As New clsEmployees
Dim recEmployee As New clsEmployee
Dim LastRow As Integer, myCount As Integer
Dim EmpArray As Variant

LastRow = ActiveSheet.Cells(ActiveSheet.Rows.Count, 1).End(xlUp).Row
EmpArray = ActiveSheet.Range(Cells(1, 1), Cells(LastRow, 4))

For myCount = 1 To UBound(EmpArray)
    Set recEmployee = New clsEmployee
    With recEmployee
        .EmpName = EmpArray(myCount, 1)
        .EmpID = EmpArray(myCount, 2)
        .EmpRate = EmpArray(myCount, 3)
        .EmpWeeklyHrs = EmpArray(myCount, 4)
        colEmployees.Add recEmployee
    End With
Next myCount

MsgBox "Número de funcionários: " & colEmployees.Count & Chr(10) & _
    "Nome do funcionário(2): " & colEmployees.Item(2).EmpName
MsgBox "Pagamento semanal de Tracy: $" & colEmployees("1651").EmpWeeklyPay

For Each recEmployee In colEmployees.Items
    recEmployee.EmpRate = recEmployee.EmpRate * 1.5
Next recEmployee

MsgBox "Pagamento semanal de Tracy (após Bônus): $" & colEmployees.Item("1651").EmpWeeklyPay

Set recEmployee = Nothing

End Sub
```

Esse programa não é diferente do que é utilizado com a coleção-padrão, mas existem algumas diferenças-chave. `colEmployees` é declarada como tipo `clsEmployees`, a nova coleção de módulo de classe, em vez de ser declarada como `Collection`. O array e a coleção são preenchidos da mesma maneira, mas o modo como os registros na coleção são referenciados mudou. Ao referenciar um membro da coleção, como o registro do funcionário 2, a propriedade `Item` deve ser utilizada. Compare a sintaxe das caixas de mensagem nesse programa com a do programa anterior.

O loop `For Each Next` passa por cada registro na coleção e multiplica o `EmpRate` por 1,5, alterando seu valor. O resultado desse 'bônus' é mostrado em uma caixa de mensagem semelhante à mostrada anteriormente na Figura 22.

Estudo de caso

Botões de ajuda

Você tem uma planilha complexa que requer um modo de o usuário obter ajuda. Você poderia colocar as informações em caixas de comentário, mas elas não são muito óbvias, especialmente para o usuário de Excel iniciante. Outra opção é criar botões de ajuda.

Na planilha, crie pequenos rótulos com um ponto de interrogação em cada um. Para obter a aparência de botão como a mostrada na Figura 22.8, configure a propriedade `SpecialEffect` dos rótulos com `Raised` e escureça a `BackColor`. Coloque um rótulo por linha. A duas colunas do botão, insira o texto de ajuda que você quer que apareça quando o rótulo for clicado. Oculte essa coluna de texto de ajuda.

Figura 22.8
Anexe os botões de ajuda à planilha e insira o texto de ajuda.

Crie um userform simples com um rótulo e um botão Fechar. Mude o nome do formulário para **`HelpForm`**, o botão para **`CloseHelp`** e o rótulo para **`HelpText`**. Dimensione o rótulo o suficiente para conter o texto de ajuda. Adicione uma macro atrás do formulário para ocultá-lo quando o botão for clicado:

```
Private Sub CloseHelp_Click()
Unload Me
End Sub
```

Insira um nome de módulo de classe **`clsLabel`**. Você precisará de uma variável, `Lbl`, para capturar os eventos de controle:

```
Public WithEvents Lbl As MSForms.Label
```

Além disso, você precisará de um método para localizar e exibir o texto de ajuda correspondente:

```
Private Sub Lbl_Click()
Dim Rng As Range

Set Rng = Lbl.TopLeftCell

If Lbl.Caption = "?" Then
    HelpForm.Caption = "Rótulo na célula " & Rng.Address(0, 0)
    HelpForm.HelpText.Caption = Rng.Offset(, 2).Value
    HelpForm.Show
End If

End Sub
```

No módulo `ThisWorkbook`, crie um procedimento `Workbook_Open` para criar uma coleção dos rótulos na pasta de trabalho:

```
Option Explicit
Option Base 1
Dim col As Collection

Sub Workbook_Open()
Dim WS As Worksheet
Dim cLbl As clsLabel
Dim OleObj As OLEObject

Set col = New Collection

For Each WS In ThisWorkbook.Worksheets
```

```
        For Each OleObj In WS.OLEObjects
            If OleObj.OLEType = xlOLEControl Then
                If TypeName(OleObj.Object) = "Rótulo" Then
                    Set cLbl = New clsLabel
                    Set cLbl.Lbl = OleObj.Object
                    col.Add cLbl
                End If
            End If
        Next OleObj
    Next WS

    End Sub
```

Execute `Workbook_Open` para criar a coleção. Clique um rótulo na planilha. O texto de ajuda correspondente aparecerá no formulário de ajuda, como mostrado na Figura 22.9.

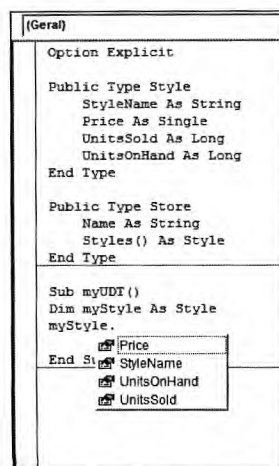
Figura 22.9
O texto de ajuda está apenas a um clique de distância.

Tipos definidos pelo usuário

Os tipos definidos pelo usuário (*user-defined types* – UDTs)fornecem um pouco do poder de um objeto personalizado, mas sem a necessidade de um módulo de classe. Um módulo de classe permite a criação de propriedades e métodos personalizados, ao passo que um UDT permite somente propriedades personalizadas. Às vezes, porém, isso é tudo de que você precisa.

Um UDT é declarado com uma instrução `Type..End Type`. Pode ser `Public` ou `Private`. O UDT recebe um nome tratado como um objeto. Dentro de `Type`, as variáveis individuais são declaradas para tornarem-se as propriedades do UDT.

Dentro de um procedimento real, uma variável é definida como sendo do tipo personalizado. Quando essa variável é utilizada, as propriedades estão disponíveis, do mesmo modo como em um objeto personalizado (veja Figura 22.10).

Figura 22.10
As propriedades de um UDT estão disponíveis como em um objeto personalizado.

O exemplo seguinte utiliza dois UDTs para resumir um relatório de estilos de produto em várias lojas.

O primeiro UDT consiste em propriedades para cada estilo de produto:

```
Option Explicit
Public Type Style
    StyleName As String
    Price As Single
    UnitsSold As Long
    UnitsOnHand As Long
End Type
```

O segundo UDT consiste no nome da loja e um array, cujo tipo é o primeiro UDT:

```
Public Type Store
    Name As String
    Styles() As Style
End Type
```

Depois que os UDTs são estabelecidos, o programa principal é escrito. Somente uma variável do segundo tipo de UDT, `Store`, é necessária porque esse tipo contém o primeiro tipo, `Style` (veja Figura 22.11). Mas todas as propriedades dos UDTs estão facilmente disponíveis. E, com o uso do UDT, as diversas variáveis podem ser lembradas facilmente — elas estão a apenas um ponto (.) de distância:

```
Sub Main()
Dim FinalRow As Integer, ThisRow As Integer, ThisStore As Integer
Dim CurrRow As Integer, TotalDollarsSold As Integer, TotalUnitsSold As Integer
Dim TotalDollarsOnHand As Integer, TotalUnitsOnHand As Integer
Dim ThisStyle As Integer
Dim StoreName As String

ReDim Stores(0 To 0) As Store ' O UDT é declarado

FinalRow = Cells(Rows.Count, 1).End(xlUp).Row

'O seguinte loop FOR preenche ambos arrays. O array externo é preenchido com o
'nome da loja e um array consistindo em detalhes de produto.
'Para realizar isso, o nome de loja é monitorado e quando muda,
'o array externo é expandido.
'O array interno para cada array externo expande a cada novo produto
Para ThisRow = 2 Para FinalRow
    StoreName = Range("A" & ThisRow).Value
'Verifica se essa é a primeira entrada no array externo
    If LBound(Stores) = 0 Then
        ThisStore = 1
        ReDim Stores(1 To 1) As Store
        Stores(1).Name = StoreName
        ReDim Stores(1).Styles(0 To 0) As Style
    Else
        For ThisStore = LBound(Stores) To UBound(Stores)
            If Stores(ThisStore).Name = StoreName Then Exit For
        Next ThisStore
        If ThisStore > UBound(Stores) Then
            ReDim Preserve Stores(LBound(Stores) To UBound(Stores) + 1) As Store
            Stores(ThisStore).Name = StoreName
            ReDim Stores(ThisStore).Styles(0 To 0) As Style
        End If
    End If
    With Stores(ThisStore)
        If LBound(.Styles) = 0 Then
            ReDim .Styles(1 To 1) As Style
        Else
            ReDim Preserve .Styles(LBound(.Styles) ToUBound(.Styles) + 1) As Style
        End If
        With .Styles(UBound(.Styles))
            .StyleName = Range("B" & ThisRow).Value
            .Price = Range("C" & ThisRow).Value
            .UnitsSold = Range("D" & ThisRow).Value
            .UnitsOnHand = Range("E" & ThisRow).Value
        End With
    End With
Next ThisRow

'Cria um relatório em uma nova planilha
Sheets.Add
Range("A1:E1").Value = Array("Nome da Loja", "Unidades Vendidas","Reais Vendidos", "Unidades Disponíveis", _
    "Reais Disponíveis")
CurrRow = 2

For ThisStore = LBound(Stores) To UBound(Stores)
    With Stores(ThisStore)
        TotalDollarsSold = 0
        TotalUnitsSold = 0
        TotalDollarsOnHand = 0
        TotalUnitsOnHand = 0
```

```
'Passa pelo array de estilos de produto dentro do array
'de lojas para resumir as informações
        For ThisStyle = LBound(.Styles) To UBound(.Styles)
            With .Styles(ThisStyle)
                TotalDollarsSold = TotalDollarsSold + .UnitsSold * .Price
                TotalUnitsSold = TotalUnitsSold + .UnitsSold
                TotalDollarsOnHand = TotalDollarsOnHand + .UnitsOnHand * .Price
                TotalUnitsOnHand = TotalUnitsOnHand + .UnitsOnHand
            End With
        Next ThisStyle
        Range("A" & CurrRow & ":E" & CurrRow).Value =Array(.Name, TotalUnitsSold, TotalDollarsSold,TotalUnitsOnHa
nd, TotalDollarsOnHand)
    End With
    CurrRow = CurrRow + 1
Next ThisStore

End Sub
```

Figura 22.11
Os UDTs podem fazer com que um programa multivariáveis potencialmente confuso se torne mais fácil de escrever. (Observação: por questão de conveniência, os resultados do programa foram combinados com os dados brutos.)

	1	2	3	4	5
1	Loja	Estilo	Preço	Unidades Vendidas	Unidades Disponíveis
2	Loja A	Estilo C	96,87	16	45
3	Loja A	Estilo A	38,43	7	94
4	Loja A	Estilo B	91,24	5	18
5	Loja A	Estilo E	19,89	0	96
6	Loja A	Estilo D	2,45	20	66
7	Loja B	Estilo B	92,59	4	83
8	Loja B	Estilo A	15,75	9	66
9	Loja B	Estilo F	13,12	2	35
10	Loja B	Estilo G	30,86	22	37
11	Loja B	Estilo H	37,38	21	77
12					
13					
14					
15					
16					
17	Nome da Loja	Unidades Vendidas	Reais Vendidos	Unidades Disponíveis	Reais Disponíveis
18	Loja A	48	R$ 2.324	319	R$ 11.684
19	Loja B	58	R$ 2.002	298	R$ 13.203
20					

Próximos passos

No Capítulo 23, "Técnicas avançadas de userform", você aprenderá mais sobre controles e técnicas para construir userforms.

Técnicas avançadas de userform

23

O Capítulo 10, "Userforms — uma introdução", abrangeu os princípios básicos de como adicionar controles a userforms. Este capítulo continuará esse tópico, examinando controles e métodos mais avançados para tirar o máximo proveito dos userforms.

NESTE CAPÍTULO

Utilizando a barra de ferramentas UserForm no Design de Controles em Userforms

No Editor do VB, oculto no menu Exibir, Barras de Ferramentas, há algumas barras de ferramentas que não aparecem a menos que haja intervenção do usuário. Uma delas é a UserForm, mostrada na Figura 23.1.

Figura 23.1
A barra de ferramentas UserForm tem ferramentas para organizar os controles em um userform.

Mais controles userform

O Capítulo 10 iniciou uma revisão de alguns controles disponíveis em userforms. A revisão continua aqui. No fim da revisão de cada controle há uma tabela relacionando os eventos desse controle.

Caixas de seleção

As caixas de seleção permitem que o usuário selecione uma ou mais opções em um userform. Diferentemente dos botões de opção que discutimos no Capítulo 10, um usuário pode selecionar uma ou mais caixas de seleção por vez.

O valor de uma caixa de seleção marcada é `True`; o valor de uma caixa de seleção desmarcada é `False`. Se tirar o valor de uma caixa de seleção (`Checkbox1.value = ""`), quando o userform executar, a caixa de seleção terá uma marca desativada, como mostrado na Figura 23.2. Isso pode ser útil para verificar se usuários visualizaram todas as opções e fizeram uma seleção.

Figura 23.2
Utilize o valor nulo da caixa de seleção para verificar se os usuários visualizaram e responderam a todas as opções.

O código a seguir revisa todas as caixas de seleção no grupo de idiomas e, se um valor for nulo, pede que o usuário revise as seleções:

```
Private Sub btnClose_Click()

Dim Msg As String
Dim Chk As Control

Set Chk = Nothing

'Restringe a pesquisa apenas aos controles da 2ª página
For Each Chk In frm_Multipage.MultiPage1.Pages(1).Controls
    'Só precisa verificar controles da caixa de seleção
    If TypeName(Chk) = "CheckBox" Then
        'e, só em caso de adicionarmos mais controles de caixa de seleção,
 'just check the ones in the group
        If Chk.Object.GroupName = "Idiomas" Then
            'se o valor for nulo (o valor de propriedade está vazio)
            If IsNull(Chk.Object.Value) Then
                'adiciona a legenda a uma string
                Msg = Msg & vbNewLine & Chk.Caption
            End If
        End If
    End If
Next Chk

If Msg <> "" Then
    Msg = "As seguintes caixas de seleção não foram verificadas:" & vbNewLine & Msg
    MsgBox Msg, vbInformation, "Exigida informação adicional"
End If

End Sub
```

A Tabela 23.1 relaciona os eventos dos controles `CheckBox`.

Tabela 23.1 Eventos dos controles CheckBox

Evento	Descrição
`AfterUpdate`	Ocorre depois que uma caixa de seleção foi marcada e/ou desmarcada.
`BeforeDragOver`	Ocorre enquanto o usuário arrasta e solta dados sobre a caixa de seleção.
`BeforeDropOrPaste`	Ocorre antes de o usuário estar prestes a soltar ou colar dados sobre a caixa de seleção.
`BeforeUpdate`	Ocorre antes de a caixa de seleção ser marcada e/ou desmarcada.
`Change`	Ocorre quando o valor da caixa de seleção é alterado.
`Click`	Ocorre quando o usuário clica no controle com o mouse.
`DblClick`	Ocorre quando o usuário dá um clique duplo na caixa de seleção com o mouse.
`Enter`	Ocorre imediatamente antes de a caixa de seleção receber o foco de outro controle no mesmo userform.
`Error`	Ocorre quando a caixa de seleção se depara com um erro e não pode retornar as informações de erro.
`Exit`	Ocorre logo depois que a caixa de seleção perde foco para outro controle no mesmo userform.
`KeyDown`	Ocorre quando o usuário pressiona uma tecla no teclado.
`KeyPress`	Ocorre quando o usuário pressiona uma tecla ANSI. Uma tecla ANSI é um caractere digitável, como a letra *A*.
`KeyUp`	Ocorre quando o usuário solta uma tecla no teclado.
`MouseDown`	Ocorre quando o usuário pressiona o botão do mouse dentro das bordas da caixa de seleção.
`MouseMove`	Ocorre quando o usuário move o mouse dentro das bordas da caixa de seleção.
`MouseUp`	Ocorre quando o usuário pressiona o botão do mouse dentro das bordas da caixa de seleção.

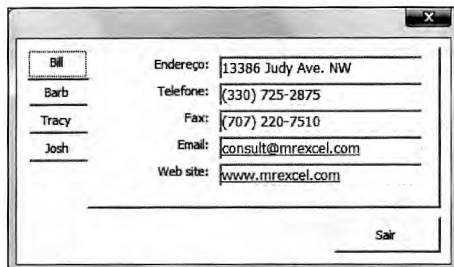
Tab Strips

O controle `MultiPage` permite que um userform tenha várias páginas. Toda página do formulário pode ter seu próprio conjunto de controles, sem nenhuma relação com qualquer outro controle no formulário. Um controle `TabStrip` também permite que um userform tenha muitas páginas, mas os controles em uma barra de guias são idênticos; são desenhados uma única vez. Apesar disso, quando o formulário é executado, as informações mudam de acordo com a barra de guias que estiver ativa (veja Figura 23.3).

Figura 23.3
Uma barra de guias permite que um userform com diversas páginas compartilhe controles, mas não informações.

➔ Para aprender mais sobre controles MultiPage, **consulte** "Utilizando o controle MultiPage para combinar formas," **p. 191**, no Capítulo 10.

Por padrão, uma barra de guias é estreita, com duas guias na parte superior. Clicar com o botão direito do mouse em uma guia permite adicionar, remover, renomear ou mover uma guia. A barra de guias também deve ser dimensionada para armazenar todos os controles. Um botão para fechar o formulário deve ser desenhado fora da área da barra de guias.

Também é possível mover as guias pela barra. Isso é feito alterando-se a propriedade `TabOrientation`. As guias podem estar na parte de cima e de baixo e do lado esquerdo ou direito do userform.

As linhas de código a seguir foram usadas para criar o formulário de barra de guias mostrado na Figura 23.3. A sub `Initialize` chama a sub `SetValuestoTabStrip`, que configura o valor para a primeira guia:

```
Private Sub UserForm_Initialize()
SetValuesToTabStrip 1 'As default
End Sub
```

Essas linhas de código tratam o que acontece quando uma nova guia é selecionada.

```
Private Sub TabStrip1_Change()
Dim lngRow As Long

lngRow = TabStrip1.Value + 1
SetValuesToTabStrip lngRow

End Sub
```

Essa sub fornece os dados mostrados em cada guia. Uma planilha foi configurada, com cada linha correspondendo a uma guia.

```
Private Sub SetValuesToTabStrip(ByVal lngRow As Long)
With frm_Staff
    .lbl_Name.Caption = Cells(lngRow, 2).Value
    .lbl_Phone.Caption = Cells(lngRow, 3).Value
    .lbl_Fax.Caption = Cells(lngRow, 4).Value
    .lbl_E-mail.Caption = Cells(lngRow, 5).Value
    .lbl_Website.Caption = Cells(lngRow, 6).Value
    .Show
End With
End Sub
```

Os valores da barra de guias são automaticamente preenchidos. Eles correspondem à posição da guia na barra; mover uma guia altera o seu valor.

> **DICA** Se você quiser que uma única guia tenha um controle extra, o controle poderia ser adicionado em tempo de execução, quando a guia tornou-se ativa, e removido quando a guia for desativada.

A Tabela 23.2 lista os eventos para o controle `TabStrip`.

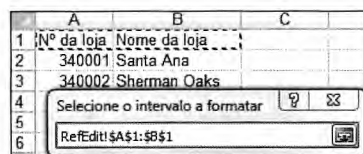
Tabela 23.2 Eventos para controles `TabStrip`

Evento	Descrição
`BeforeDragOver`	Ocorre enquanto o usuário arrasta e solta dados sobre o controle.
`BeforeDropOrPaste`	Ocorre imediatamente antes de o usuário soltar ou colar dados no controle.
`Change`	Ocorre quando o valor do controle é alterado.
`Click`	Ocorre quando o usuário clica no controle com o mouse.
`DblClick`	Ocorre quando o usuário dá um clique duplo no controle com o mouse.
`Enter`	Ocorre antes de o controle receber o foco de outro controle no mesmo userform.
`Error`	Ocorre quando o controle encontra um erro e não pode retornar as informações sobre o erro.
`Exit`	Ocorre assim que o foco do controle passa para outro controle no mesmo userform.
`KeyDown`	Ocorre quando o usuário pressiona uma tecla no teclado.
`KeyPress`	Ocorre quando o usuário pressiona uma tecla ANSI. Uma tecla ANSI é um caractere digitável, como a letra *A*.
`KeyUp`	Ocorre quando o usuário solta uma tecla no teclado.
`MouseDown`	Ocorre quando o usuário pressiona o botão do mouse dentro das bordas do controle.
`MouseMove`	Ocorre quando o usuário move o mouse dentro das bordas do controle.
`MouseUp`	Ocorre quando o usuário libera o botão do mouse dentro das bordas do controle.

`RefEdit`

O controle `RefEdit` permite que o usuário selecione um intervalo em uma planilha; o intervalo é retornado como o valor do controle. Ele pode ser adicionado a qualquer formulário. Uma vez ativado por um clique do botão no lado direito do campo, o userform desaparece e é substituído pelo formulário de seleção de intervalo, utilizado ao selecionar intervalos com muitas ferramentas do assistente do Excel. Clique no botão à direita para mostrar o userform mais uma vez.

O formulário na Figura 23.4 e o código a seguir permitem que o usuário selecione um intervalo, que então se torna negrito.

Figura 23.4 Utilize `RefEdit` para permitir que o usuário selecione um intervalo em uma planilha.

```
Private Sub cb1_Click()
Range(RefEdit1.Value).Font.Bold = True
End Sub
```

A Tabela 23.3 relaciona os eventos dos controles `RefEdit`.

Tabela 23.3 Eventos para controles `RefEdit`

Evento	Descrição
`AfterUpdate`	Ocorre depois que os dados do controle foram alterados pelo usuário.
`BeforeDragOver`	Ocorre enquanto o usuário arrasta e solta dados sobre o controle.
`BeforeDropOrPaste`	Ocorre imediatamente antes de o usuário soltar ou colar dados no controle.
`BeforeUpdate`	Ocorre antes de os dados no controle serem alterados.
`Change`	Ocorre quando o valor do controle é alterado.

Evento	Descrição
Click	Ocorre quando o usuário clica no controle com o mouse.
DblClick	Ocorre quando o usuário dá um clique duplo no controle com o mouse.
Enter	Ocorre quando a lista suspensa aparece ao se pressionar a seta suspensa da caixa de combinação ou F4 no teclado.
Error	Ocorre antes de o controle receber o foco de outro controle no mesmo userform.
Exit	Ocorre quando o controle encontra um erro e não pode retornar as informações sobre o erro.
KeyDown	Ocorre assim que o foco do controle passa para outro controle no mesmo userform.
KeyPress	Ocorre quando o usuário pressiona uma tecla no teclado.
KeyUp	Ocorre quando o usuário pressiona uma tecla ANSI. Uma tecla ANSI é um caractere digitável, como a letra *A*.
MouseDown	Ocorre quando o usuário solta uma tecla no teclado.
MouseMove	Ocorre quando o usuário pressiona o botão do mouse dentro das bordas do controle.
MouseUp	Ocorre quando o usuário move o mouse dentro das bordas do controle.
MouseUp	Ocorre quando o usuário libera o botão do mouse dentro das bordas do controle.

Botões alternadores

Um botão alternador parece um botão de comando comum, mas quando o usuário o pressiona, ele permanece pressionado até ser novamente selecionado. Isso permite que um valor `True` ou `False` seja retornado com base no status do botão. A Tabela 23.4 relaciona os eventos dos controles `ToggleButton`.

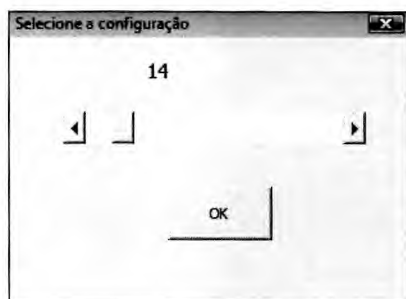
Tabela 23.4 Eventos dos controles `ToggleButton`

Evento	Descrição
AfterUpdate	Ocorre depois que os dados do controle foram alterados pelo usuário.
BeforeDragOver	Ocorre enquanto o usuário arrasta e solta dados sobre o controle.
BeforeDropOrPaste	Ocorre imediatamente antes de o usuário soltar ou colar dados no controle.
BeforeUpdate	Ocorre antes de os dados no controle serem alterados.
Change	Ocorre quando o valor do controle é alterado.
Click	Ocorre quando o usuário clica no controle com o mouse.
DblClick	Ocorre quando o usuário dá um clique duplo no controle com o mouse.
Enter	Ocorre antes de o controle receber o foco de outro controle no mesmo userform.
Error	Ocorre quando o controle encontra um erro e não pode retornar as informações sobre o erro.
Exit	Ocorre assim que o foco do controle passa para outro controle no mesmo userform.
KeyDown	Ocorre quando o usuário pressiona uma tecla no teclado.
KeyPress	Ocorre quando o usuário pressiona uma tecla ANSI. Uma tecla ANSI é um caractere digitável, como a letra *A*.
KeyUp	Ocorre quando o usuário solta uma tecla no teclado.
MouseDown	Ocorre quando o usuário pressiona o botão do mouse dentro das bordas do controle.
MouseMove	Ocorre quando o usuário move o mouse dentro das bordas do controle.
MouseUp	Ocorre quando o usuário libera o botão do mouse dentro das bordas do controle.

Utilizando uma barra de rolagem como um controle deslizante para selecionar valores

O Capítulo 10 discutiu como utilizar um controle `SpinButton` para permitir que alguém escolha uma data. O botão spin é bom, mas permite que os clientes ajustem para cima ou para baixo apenas uma unidade por vez. Um método alternativo é desenhar uma barra de rolagem horizontal no meio do userform e utilizá-la como um controle deslizante. Os clientes podem utilizar setas nas extremidades da barra de rolagem como as setas de botão spin, mas também podem pegar a barra de rolagem e arrastá-la instantaneamente para certo valor.

O userform mostrado na Figura 23.5 inclui um rótulo chamado Label1 e uma barra de rolagem chamada ScrollBar1.

Figura 23.5 Utilizar um controle de barra de rolagem permite que o usuário arraste rapidamente a barra para um valor numérico ou de dados específico.

O código `Initialize` do userform configura os valores `Min` e `Max` da barra de rolagem. Ele inicializa a barra de rolagem para um valor da célula A1 e atualiza o `Label1.Caption`:

```
Private Sub UserForm_Initialize()
    Me.ScrollBar1.Min = 0
    Me.ScrollBar1.Max = 100
    Me.ScrollBar1.Value = Range("A1").Value
    Me.Label1.Caption = Me.ScrollBar1.Value
End Sub
```

Duas rotinas de tratamento de evento são necessárias para a barra de rolagem. O evento `Change` trata se usuários clicam nas setas nas extremidades da barra de rolagem. O evento Scroll trata se eles arrastam o controle deslizante para um novo valor:

```
Private Sub ScrollBar1_Change()
    ' Esse evento trata se usuários tocam
    ' nas setas na extremidade da barra de rolagem
    Me.Label1.Caption = Me.ScrollBar1.Value
End Sub

Private Sub ScrollBar1_Scroll()
    ' Esse evento trata se eles arrastam o controle deslizante
    Me.Label1.Caption = Me.ScrollBar1.Value
End Sub
```

Por fim, o evento anexado ao botão para gravar o valor da barra de rolagem na planilha:

```
Private Sub CommandButton1_Click()
    Range("A1").Value = Me.ScrollBar1.Value
    Unload Me
End Sub
```

A Tabela 23.5 relaciona os eventos dos controles `Scrollbar`.

Tabela 23.5 Eventos dos controles `Scrollbar`

Evento	Descrição
`AfterUpdate`	Ocorre depois que os dados do controle foram alterados pelo usuário.
`BeforeDragOver`	Ocorre enquanto o usuário arrasta e solta dados sobre o controle.
`BeforeDropOrPaste`	Ocorre imediatamente antes de o usuário soltar ou colar dados no controle.
`BeforeUpdate`	Ocorre antes de os dados no controle serem alterados.

Evento	Descrição
`Change`	Ocorre quando o valor do controle é alterado.
`Click`	Ocorre quando o usuário clica no controle com o mouse.
`DblClick`	Ocorre quando o usuário dá um clique duplo no controle com o mouse.
`DropButtonClick`	Ocorre quando a lista suspensa aparece ao se pressionar a seta suspensa da caixa de combinação ou F4 no teclado.
`Enter`	Ocorre antes de o controle receber o foco de outro controle no mesmo userform.
`Error`	Ocorre quando o controle encontra um erro e não pode retornar as informações sobre o erro.
`Exit`	Ocorre assim que o foco do controle passa para outro controle no mesmo userform.
`KeyDown`	Ocorre quando o usuário pressiona uma tecla no teclado.
`KeyPress`	Ocorre quando o usuário pressiona uma tecla ANSI. Uma tecla ANSI é um caractere digitável, como a letra *A*.
`KeyUp`	Ocorre quando o usuário solta uma tecla no teclado.
`MouseDown`	Ocorre quando o usuário pressiona o botão do mouse dentro das bordas do controle.
`MouseMove`	Ocorre quando o usuário move o mouse dentro das bordas do controle.
`MouseUp`	Ocorre quando o usuário libera o botão do mouse dentro das bordas do controle.

Controles e coleções

No Capítulo 22, "Criando classes, registros e coleções", vários rótulos em uma planilha são agrupados em uma coleção. Com um pouco mais de código, esses rótulos transformaram-se em telas de ajuda para os usuários. Os controles de userform também podem ser agrupados em coleções e tiram proveito de módulos de classe.

O seguinte exemplo marca ou desmarca todas as caixas de seleção no userform, dependendo do rótulo que o usuário escolher.

Coloque o seguinte código no módulo de classe, `clsFormEvents`. Consiste em uma propriedade, `chb`, e dois métodos, `SelectAll` e `UnselectAll`.

O método `SelectAll` coloca uma marca de verificação em uma caixa de seleção, configurando seu valor como `True`:

```
Option Explicit
Public WithEvents chb As MSForms.CheckBox

Public Sub SelectAll()
chb.Value = True
End Sub
```

O método UnselectAll remove a marca da caixa de seleção:

```
Public Sub UnselectAll()
chb.Value = False
End Sub
```

Esse método configura o módulo de classe. Em seguida, os controles precisam ser colocados em uma coleção. O código a seguir, colocado atrás do formulário, `frm_Movies`, coloca as caixas de seleção em uma coleção. Essas caixas fazem parte de um frame, `f_Selection`, que facilita a criação da coleção porque restringe o número de controles que precisam ser marcados por todo userform apenas àqueles dentro do frame:

```
Option Explicit
Dim col_Selection As New Collection

Private Sub UserForm_Initialize()
Dim ctl As MSForms.CheckBox
Dim chb_ctl As clsFormEvents

'Itera pelos membros do frame e os adiciona à coleção
For Each ctl In f_Selection.Controls
    Set chb_ctl = New clsFormEvents
    Set chb_ctl.chb = ctl
    col_Selection.Add chb_ctl
Next ctl
```

```
End Sub
```

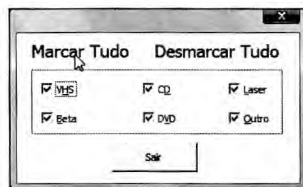
Quando o formulário é aberto, os controles são colocados na coleção. Tudo que resta a fazer agora é adicionar o código dos rótulos a fim de selecionar e remover a seleção das caixas de seleção:

```
Private Sub lbl_SelectAll_Click()
Dim ctl As clsFormEvents

For Each ctl In col_Selection
    ctl.SelectAll
Next ctl

End Sub
```

O código a seguir remove a seleção das caixas de seleção na coleção:

```
Private Sub lbl_unSelectAll_Click()
Dim ctl As clsFormEvents

For Each ctl In col_Selection
    ctl.Unselectall
Next ctl

End Sub
```

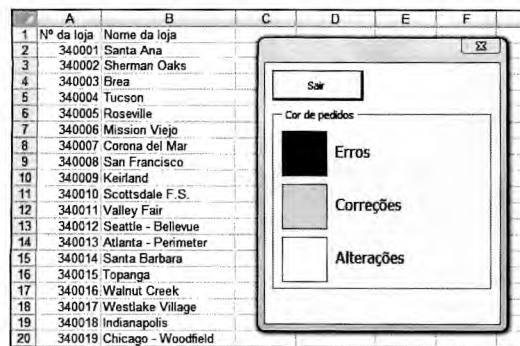
Todas as caixas de seleção podem ser marcadas e desmarcadas com um único clique do mouse, como mostrado na Figura 23.6.

Figura 23.6
Utilize frames, coleções e módulos de classe em conjunto para criar userforms rápidos e eficientes.

> **DICA**
> Se seus controles não podem ser colocados em um frame, utilize um `tag` para criar um agrupamento improvisado. Um `tag` é uma propriedade que armazena mais informações sobre um controle. Seu valor é do tipo `string`, que pode armazenar qualquer tipo de informação. Por exemplo, um `tag` pode ser utilizado para criar um grupo informal de controles de agrupamentos diferentes.

Userforms amodais

Já teve um userform ativo mas precisou ver algo em uma planilha? Houve uma época em que o formulário tinha de ser desativado antes que qualquer outra coisa pudesse ser feita no Excel. Hoje não mais! Os formulários agora podem ser amodais, o que significa que não têm de interferir com a funcionalidade do Excel. O usuário pode digitar em uma célula, alternar entre outra planilha, copiar/colar dados e utilizar barras de ferramentas e menus — é como se o userform não estivesse presente.

Por padrão, um userform é modal, o que significa que não há nenhuma outra interação com o Excel a não ser o formulário. Para tornar o formulário amodal, mude a propriedade `ShowModal` para `False`. Depois de definido como amodal, o usuário pode selecionar uma célula na planilha enquanto o formulário está ativo, como mostrado na Figura 23.7.

Figura 23.7
Um formulário amodal permite que o usuário insira uma célula enquanto o formulário ainda está ativo.

Utilizando hiperlinks em userforms

No exemplo de userform mostrado na Figura 23.3, há um campo para e-mail e um endereço de site Web. Não seria interessante se, ao clicar nesses itens, uma mensagem de e-mail em branco ou uma página Web aparecesse automaticamente? Você pode! O programa a seguir cria uma nova mensagem ou abre um navegador da Web quando o rótulo correspondente é clicado.

A declaração da interface de programas aplicativos (*applications programming interface* – API) e quaisquer outras constantes, vão na parte superior do código.

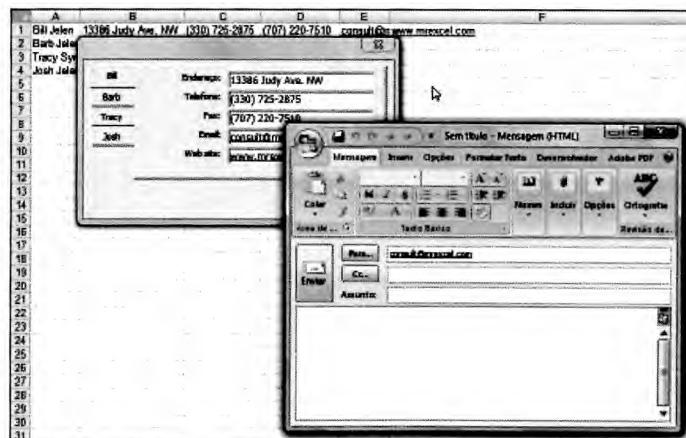
```
Private Declare Function ShellExecute Lib "shell32.dll" Alias"ShellExecuteA"(ByVal hWnd As Long, ByVal
lpOperation As String,ByVal lpFile As String, ByVal lpParameters As String,ByVal lpDirectory As String, ByVal
nShowCmd As Long) As Long
Const SWNormal = 1
```

Esta sub controla o que acontece quando o rótulo de correio eletrônico é clicado, como mostrado na Figura 23.8:

```
Private Sub lbl_E-mail_Click()
Dim lngRow As Long

lngRow = TabStrip1.Value + 1
ShellExecute 0&, "open", "mailto:" & Cells(lngRow, 5).Value,vbNullString, vbNullString, SWNormal

End Sub
```

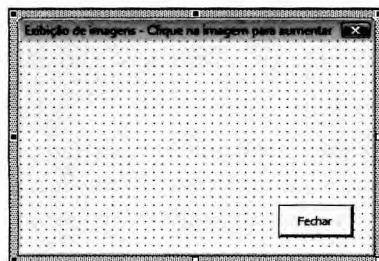
Esta sub controla o que acontece quando o rótulo de site é clicado:

```
Private Sub lbl_Website_Click()
Dim lngRow As Long

lngRow = TabStrip1.Value + 1
ShellExecute 0&, "open", Cells(lngRow, 6).Value, vbNullString,vbNullString, SWNormal

End Sub
```

Figura 23.8
Transforme os endereços de correio eletrônico e de sites Web em links clicáveis.

Adicionando controles em tempo de execução

É possível adicionar controles a um userform em tempo de execução. Isso é conveniente se você não tiver certeza de quantos itens adicionará ao formulário.

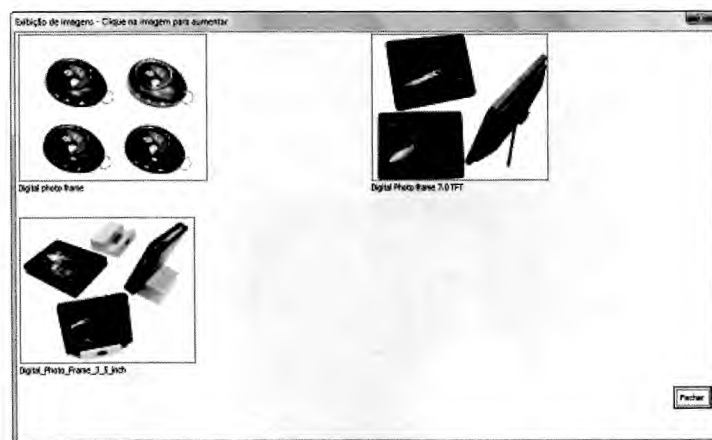
A Figura 23.9 mostra um formulário simples. Você tem apenas um botão. Esse formulário simples é utilizado para exibir qualquer número de imagens em um catálogo de produtos. As figuras e seus respectivos rótulos aparecem em tempo de execução, enquanto o formulário estiver sendo exibido.

Figura 23.9
Formulários flexíveis podem ser criados se você adicionar a maioria de controles em tempo de execução.

Um representante de vendas, ao fazer uma apresentação, utiliza esse formulário para exibir o catálogo de produtos. Você pode selecionar qualquer número de itens de estoque de uma planilha Excel e pressionar uma tecla de atalho para exibir o formulário. Se você selecionar 20 itens na planilha, o formulário será exibido com todas as figuras relativamente pequenas, como mostrado na Figura 23.10.

Se o representante de vendas selecionar menos itens, as imagens serão mostradas em um formato maior, como visto na Figura 23.11.

Várias técnicas são utilizadas para criar esse userform instantaneamente. O formulário inicial contém apenas um botão, chamado `cbClose`. Qualquer outra coisa pode ser adicionada durante a apresentação.

Figura 23.10
Aqui, o representante de vendas solicitou a visualização de fotos de 20 itens de estoque. O procedimento `UserForm_Initialize` adiciona todas as figuras e rótulos instantaneamente.

Figura 23.11
A lógica `inUserform_Initialize` decide quantas figuras serão exibidas e adiciona os controles de tamanho apropriados.

Redimensionando o userform instantaneamente

Um objetivo é fornecer a melhor visualização das imagens no catálogo de produtos. Isso significa fazer o formulário aparecer com o maior tamanho possível. O código a seguir utiliza as propriedades `Height` e `Width` do formulário para confirmar que este ocupa quase toda a tela:

```
' Redimensiona o formulário
Me.Height = Int(0.98 * ActiveWindow.Height)
Me.Width = Int(0.98 * ActiveWindow.Width)
```

Adicionando um controle instantaneamente

Para um controle normal adicionado em tempo de projeto, é fácil referir-se a ele utilizando seu nome:

```
Me.cbSave.Left = 100
```

Mas, para um controle que é adicionado em tempo de execução, você tem de utilizar a coleção `Controls` para definir qualquer propriedade para ele. Portanto, é importante definir uma variável para armazenar o nome do controle. Os controles são adicionados com o método `.Add`. Um parâmetro importante é o `bstrProgId`. Esse nome de código dita se o controle adicionado é um rótulo, uma caixa de texto, um botão de comando ou qualquer outra coisa.

O próximo código adiciona um novo rótulo ao formulário. `PicCount` é um variável contadora utilizada para assegurar que cada rótulo tenha um novo nome. Depois que o formulário é adicionado, especifique uma posição para o controle configurando as propriedades `Top` e `Left`. Você também deve configurar as propriedades `Height` e `Width` para o controle:

```
LC = "LabelA" & PicCount
Me.Controls.Add bstrProgId:="forms.label.1", Name:=LC, Visible:=True
Me.Controls(LC).Top = 25
Me.Controls(LC).Left = 50
Me.Controls(LC).Height = 18
Me.Controls(LC).Width = 60
Me.Controls(LC).Caption = cell.value
```

ATENÇÃO

Você perde algumas opções de AutoCompletar com esse método. Normalmente, se você começasse a digitar `Me.cbClose.`, as opções AutoCompletar apresentariam as escolhas válidas para um botão de comando. Mas quando você utiliza a coleção `Me.Controls(LC)` para adicionar controles instantaneamente, o VBA não sabe que tipo de controle é referenciado. Nesse caso, é útil saber que você precisa configurar a propriedade `Caption` em vez da propriedade `Value` para um *rótulo*.

Dimensionando instantaneamente

Na realidade, você precisa ser capaz de calcular valores para `Top`, `Left`, `Height` e `Width` instantaneamente. Você pode fazer isso baseando-se na altura e largura reais do formulário e na quantidade necessária de controles.

Adicionando outros controles

Para adicionar outros tipos de controles, mude o `ProgId` utilizado com o método `Add`. A Tabela 23.6 mostra os `ProgIds` para vários tipos de controles.

Tabela 23.6 Controles de userform e `ProgIds` correspondente

Controle	Identificação do programa
CheckBox	Forms.CheckBox.1
ComboBox	Forms.ComboBox.1
CommandButton	Forms.CommandButton.1
Frame	Forms.Frame.1
Image	Forms.Image.1
Label	Forms.Label.1
ListBox	Forms.ListBox.1
MultiPage	Forms.MultiPage.1
OptionButton	Forms.OptionButton.1
ScrollBar	Forms.ScrollBar.1
SpinButton	Forms.SpinButton.1
TabStrip	Forms.TabStrip.1
TextBox	Forms.TextBox.1
ToggleButton	Forms.ToggleButton.1

Adicionando uma imagem instantaneamente

Há alguma imprevisibilidade em adicionar imagens. Qualquer imagem pode ser definida como paisagem ou retrato e, talvez, seja pequena ou enorme. Uma estratégia para adicionar uma imagem é deixá-la carregar no tamanho integral, tendo antes configurado o parâmetro `.AutoSize` como `True`:

```
TC = "Image" & PicCount
Me.Controls.Add bstrProgId:="forms.image.1", Name:=TC, Visible:=True
Me.Controls(TC).Top = LastTop
Me.Controls(TC).Left = LastLeft
Me.Controls(TC).AutoSize = True
On Error Resume Next
Me.Controls(TC).Picture = LoadPicture(fname)
On Error GoTo 0
```

Então, depois de carregar a imagem, você pode ler as propriedades `Height` e `Width` do controle para determinar se a imagem é paisagem ou retrato e se ela está restringida pela largura ou altura disponível:

```
'A figura redimensionou o controle ao tamanho integral
'Determina o tamanho da figura
Wid = Me.Controls(TC).Width
Ht = Me.Controls(TC).Height
WidRedux = CellWid / Wid
HtRedux = CellHt / Ht
If WidRedux < HtRedux Then
    Redux = WidRedux
Else
    Redux = HtRedux
End If
NewHt = Int(Ht * Redux)
NewWid = Int(Wid * Redux)
```

Depois de descobrir o tamanho adequado que a imagem terá sem distorção, você pode configurar a propriedade `AutoSize` como `False` e utilizar a largura e a altura corretas para que a imagem não apareça distorcida:

```
'Agora redimensiona o controle
Me.Controls(TC).AutoSize = False
Me.Controls(TC).Height = NewHt
Me.Controls(TC).Width = NewWid
Me.Controls(TC).PictureSizeMode = fmPictureSizeModeStretch
```

Agrupando tudo

Esse é o código completo para o userform Catálogo de Imagens:

```
Private Sub UserForm_Initialize()
    ' Exibe imagens de cada código de produto selecionado na planilha
    ' Pode estar em qualquer lugar de 1 a 36 imagens
    PicPath = "C:\qimage\qi"
    Dim Pics ()

    ' Redimensiona o formulário
    Me.Height = Int(0.98 * ActiveWindow.Height)
    Me.Width = Int(0.98 * ActiveWindow.Width)

    ' Determina quantas células são selecionadas
    ' Precisamos de uma única imagem e do rótulo para cada célula
    CellCount = Selection.Cells.Count
    ReDim Preserve Pics(1 To CellCount)

    ' Descobre o tamanho do formulário redimensionado
    TempHt = Me.Height
    TempWid = Me.Width

    ' O número de colunas é um roundup de SQRT(CellCount)
    ' Assegurará 4 linhas de 5 imagens para 20 etc.
    NumCol = Int(0.99 + Sqr(CellCount))
    NumRow = Int(0.99 + CellCount / NumCol)

    ' Descobre a altura e largura de cada quadrado
    ' Cada coluna terá 2 pontos para esquerda & direita das figuras
    CellWid = Application.WorksheetFunction.Max(Int(TempWid / NumCol) - 4, 1)
    ' Cada linha precisa ter 33 pontos abaixo dele para o rótulo
    CellHt = Application.WorksheetFunction.Max(Int(TempHt / NumRow) - 33, 1)

    PicCount = 0 ' Variável contadora
    LastTop = 2
    MaxBottom = 1
    ' Constrói cada linha no formulário
```

```
    For x = 1 To NumRow
        LastLeft = 3
        ' Constrói cada coluna nessa linha
        For Y = 1 To NumCol
            PicCount = PicCount + 1
            If PicCount > CellCount Then
                ' Não há um número par de imagens para preencher
                ' fora da última linha
                Me.Height = MaxBottom + 100
                Me.cbClose.Top = MaxBottom + 25
                Me.cbClose.Left = Me.Width - 70
                Repaint
                Exit Sub
            End If
            ThisStyle = Selection.Cells(PicCount).Value
            ThisDesc = Selection.Cells(PicCount).Offset(0, 1).Value
            fname = PicPath & ThisStyle & ".jpg"
            TC = "Image" & PicCount
            Me.Controls.Add bstrProgId:="forms.image.1", Name:=TC, Visible:=True
            Me.Controls(TC).Top = LastTop
            Me.Controls(TC).Left = LastLeft
            Me.Controls(TC).AutoSize = True
            On Error Resume Next
            Me.Controls(TC).Picture = LoadPicture(fname)
            On Error GoTo 0

            ' A figura redimensionou o controle para o tamanho integral
            ' Determina o tamanho da figura
            Wid = Me.Controls(TC).Width
            Ht = Me.Controls(TC).Height
            WidRedux = CellWid / Wid
            HtRedux = CellHt / Ht
            If WidRedux < HtRedux Then
                Redux = WidRedux
            Else
                Redux = HtRedux
            End If
            NewHt = Int(Ht * Redux)
            NewWid = Int(Wid * Redux)

            ' Agora redimensiona o controle
            Me.Controls(TC).AutoSize = False
            Me.Controls(TC).Height = NewHt
            Me.Controls(TC).Width = NewWid
            Me.Controls(TC).PictureSizeMode = fmPictureSizeModeStretch
            Me.Controls(TC).ControlTipText = "Style " &ThisStyle & " " & ThisDesc

            ' Verifica a parte inferior e direita da imagem
            ThisRight = Me.Controls(TC).Left + Me.Controls(TC).Width
            ThisBottom = Me.Controls(TC).Top + Me.Controls(TC).Height
            If ThisBottom > MaxBottom Then MaxBottom = ThisBottom

            ' Adicione um rótulo abaixo da imagem
            LC = "LabelA" & PicCount
            Me.Controls.Add bstrProgId:="forms.label.1", Name:=LC, Visible:=True
            Me.Controls(LC).Top = ThisBottom + 1
            Me.Controls(LC).Left = LastLeft
            Me.Controls(LC).Height = 18
            Me.Controls(LC).Width = CellWid
            Me.Controls(LC).Caption = "Style " & ThisStyle & " " & ThisDesc

            ' Verifica onde a próxima imagem deve aparecer
            LastLeft = LastLeft + CellWid + 4
        Next Y ' Fim dessa linha
        LastTop = MaxBottom + 21 + 16
    Next x

    Me.Height = MaxBottom + 100
    Me.cbClose.Top = MaxBottom + 25
    Me.cbClose.Left = Me.Width - 70
    Repaint
End Sub
```

Adicionando ajuda ao userform

Você desenhou um excelente userform, mas agora está faltando apenas uma coisa — a instrução para os usuários. As seções a seguir mostram quatro maneiras de ajudar os usuários a preencher o formulário adequadamente.

Mostrando as teclas aceleradoras

Em geral, os formulários predefinidos têm atalhos pelo teclado que permitem que ações sejam desencadeadas ou que campos sejam selecionados com alguns pressionamentos de tecla. Esses atalhos são identificados por uma letra sublinhada em um botão ou rótulo.

Você pode adicionar essa mesma capacidade para personalizar userforms inserindo um valor na propriedade `Accelerator` do controle. Alt + tecla aceleradora seleciona o controle.

Por exemplo, na Figura 23.12, Alt+H marca a caixa de seleção VHS. Repetir a combinação desmarca a caixa.

Figura 23.12
Utilize combinações de teclas aceleradoras para fornecer aos userforms o poder dos atalhos pelo teclado.

Adicionando texto de dica de controle

Quando se passa um cursor sobre uma barra de ferramentas, o texto de dica aparece, sugerindo o que o controle faz. Você também pode adicionar o texto de dica para userforms inserindo um valor na propriedade `ControlTipText` de um controle. Na Figura 23.13, o texto de dica foi adicionado ao frame que cerca as várias categorias.

Figura 23.13
Adicione dicas aos controles para ajudar os usuários.

Criando a ordem de tabulação

Os usuários também podem navegar com a tecla Tab de um campo para outro. Esse é um recurso automático em um formulário. Para controlar para que campo a próxima tabulação leva um usuário, você pode configurar o valor da propriedade `TapStop` para cada controle.

A primeira tabulação é zero e a última é igual ao número de controles em um grupo. Lembre-se de que um grupo pode ser criado com um frame. O Excel não permite que vários controles tenham a mesma parada de tabulação. Depois de determinar as paradas de tabulação, o usuário pode utilizar a tecla Tab e a barra de espaço para selecionar e/ou remover a seleção de várias opções, como mostrado na Figura 23.14.

Figura 23.14
As opções nesse formulário foram selecionadas com a tecla Tab e a barra de espaço.

Colorindo o controle ativo

Outro método para ajudar um usuário a preencher um formulário é colorir o campo ativo. O próximo exemplo muda a cor de uma caixa de texto ou caixa de combinação quando ela está ativa.

Coloque o código seguinte em um módulo de classe chamado `clsCtlColor`:

```
Public Event GetFocus()
Public Event LostFocus(ByVal strCtrl As String)
Private strPreCtr As String
Public Sub CheckActiveCtrl(objForm As MSForms.UserForm)

With objForm
    If TypeName(.ActiveControl) = "ComboBox" OrTypeName(.ActiveControl) = "TextBox" Then
        strPreCtr = .ActiveControl.Name
        On Error GoTo Terminate
        Do
            DoEvents
            If .ActiveControl.Name <> strPreCtr Then
                If TypeName(.ActiveControl) = "ComboBox" OrTypeName(.ActiveControl) = "TextBox" Then
                    RaiseEvent LostFocus(strPreCtr)
                    strPreCtr = .ActiveControl.Name
                    RaiseEvent GetFocus
                End If
            End If
        Loop
    End If
End With

Terminate:
    Exit Sub

End Sub
```

Coloque o código seguinte atrás do userform:

```
Private WithEvents objForm As clsCtlColor

Private Sub UserForm_Initialize()
Set objForm = New clsCtlColor
End Sub
```

Esta sub altera `BackColor` do controle ativo quando o formulário é ativado:

```
Private Sub UserForm_Activate()
If TypeName(ActiveControl) = "ComboBox" OrTypeName(ActiveControl) = "TextBox" Then
   ActiveControl.BackColor = &HC0E0FF
End If
objForm.CheckActiveCtrl Me
End Sub
```

Esta sub altera `BackColor` do controle ativo quando obtém o foco:

```
Private Sub objForm_GetFocus()
ActiveControl.BackColor = &HC0E0FF
End Sub
```

Esta sub muda `BackColor` novamente para branco quando o controle perde o foco:

```
Private Sub objForm_LostFocus(ByVal strCtrl As String)
Me.Controls(strCtrl).BackColor = &HFFFFFF
End Sub
```

Esta sub limpa o `objForm` quando o formulário é fechado:

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
Set objForm = Nothing
End Sub
```

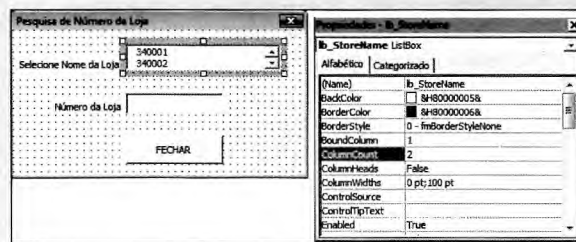
Estudo de caso

Caixas de listagem de múltiplas colunas

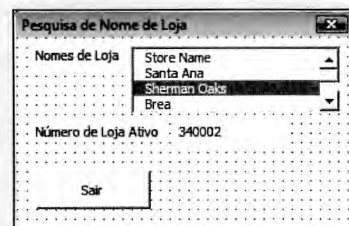
Você criou várias planilhas contendo dados. A chave primária de cada conjunto é o número de loja. A pasta de trabalho é utilizada por várias pessoas, mas nem todo mundo memoriza as lojas pelos seus respectivos números. Você precisa de alguma maneira de permitir que os usuários selecionem uma loja por nome, mas, ao mesmo tempo, retornar o número de loja a ser utilizado no código. Você poderia utilizar `PROCV` ou `CORRESP`, mas há outra maneira.

Uma caixa de listagem pode ter mais de uma coluna, mas nem todas as colunas precisam ser visíveis ao usuário. Além disso, o usuário pode selecionar um item da lista visível, mas a caixa de listagem retorna o valor correspondente de outra coluna.

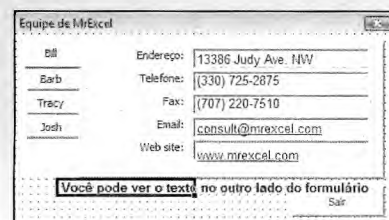
Desenhe uma caixa de listagem e configure a propriedade `ColumnCount` como `2`. Configure o `RowSource` como um intervalo de duas colunas chamado `Stores`. A primeira coluna do intervalo é o número da loja; a segunda coluna é o nome da loja. A essa altura, a caixa de listagem está exibindo ambas as colunas de dados. Para alterar isso, configure a largura da coluna como `0, 20` — o texto se atualiza automaticamente para 0 pt;20 pt. A primeira coluna agora está oculta. A Figura 23.15 mostra as propriedades de caixa de listagem como elas precisam ser.

Figura 23.15
Configurar as propriedades da caixa de listagem cria uma caixa de listagem de duas colunas que parece ter uma única coluna de dados.

A aparência da caixa de listagem agora está definida. Quando o usuário ativar a caixa de listagem, ele verá apenas os nomes de loja. Para retornar o valor da primeira coluna, configure a propriedade `BoundColumn` como `1`. Isso pode ser feito por meio da janela Propriedades ou por código. Esse exemplo utiliza o código para manter a flexibilidade de retornar o número da loja (veja Figura 23.16):

```
Private Sub UserForm_Initialize()
    lb_StoreName.BoundColumn = 1
End Sub

Private Sub lb_StoreName_Click()
lbl_StoreNum.Caption = lb_StoreName.Value
End Sub
```

Figura 23.16
Utilize uma caixa de listagem de duas colunas para permitir que o usuário selecione o nome de uma loja, mas retorne o número da loja.

Formulários transparentes

Alguma vez você já teve de tirar continuamente um formulário da frente para poder ver os dados atrás dele? O próximo código configura o userform em uma transparência de 50 por cento (veja Figura 23.17) de modo que se possam ver os dados atrás dele, sem ter de mover o formulário para outro lugar na tela (e bloquear mais dados).

Figura 23.17
Crie um formulário 50 por cento transparente para visualizar na planilha os dados atrás dele.

Coloque o seguinte código na seção de declarações do userform:

```
Private Declare Function GetActiveWindow Lib "USER32" () As Long
Private Declare Function SetWindowLong Lib "USER32" Alias"SetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As
    Long,ByVal dwNewLong As Long) As Long
Private Declare Function GetWindowLong Lib "USER32" Alias"GetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As
    Long) As Long
Private Declare Function SetLayeredWindowAttributes Lib "USER32"(ByVal hWnd As Long, ByVal crKey As Integer,
    ByVal bAlpha As Integer, ByVal dwFlags As Long) As Long
Private Const WS_EX_LAYERED = &H80000
Private Const LWA_COLORKEY = &H1
Private Const LWA_ALPHA = &H2
Private Const GWL_EXSTYLE = &HFFEC
Dim hWnd As Long
```

Coloque o seguinte código atrás de um userform. Quando o formulário for ativado, a transparência será definida:

```
Private Sub UserForm_Activate()
Dim nIndex As Long

hWnd = GetActiveWindow
nIndex = GetWindowLong(hWnd, GWL_EXSTYLE)
SetWindowLong hWnd, GWL_EXSTYLE, nIndex Or WS_EX_LAYERED
'50% semitransparente
SetLayeredWindowAttributes hWnd, 0, (255 * 50) / 100, LWA_ALPHA

End Sub
```

Próximos passos

No Capítulo 24, “Interface de programas aplicativos do Windows”, você aprenderá a acessar funções e procedimentos ocultos em arquivos no seu computador.

Interface de programas aplicativos do Windows

24

NESTE CAPÍTULO

O que é a API (*applications programming interface*) do Windows?

Apesar de todas as coisas maravilhosas que você pode fazer no Excel VBA, algumas estão fora do alcance dele e outras são simplesmente difíceis de serem feitas — como descobrir qual é a configuração da resolução de tela do usuário. É aqui que a interface de programas aplicativos Windows, ou API, pode ajudar.

Se examinar a pasta \Winnt\System32 (sistemas Windows NT), você verá uma grande quantidade de arquivos com a extensão .dll. Esses arquivos são bibliotecas de vínculos dinâmicos; contêm várias funções e procedimentos que outros programas, incluindo o VBA, podem acessar. Eles fornecem acesso de usuário a funcionalidades utilizadas pelo sistema operacional Windows e por muitos outros programas. Tenha em mente que declarações da API Windows só são acessíveis em computadores que executam o sistema operacional Microsoft Windows.

Este capítulo não ensina como escrever declarações de API, mas os princípios básicos para sua interpretação e uso. Vários exemplos úteis também foram incluídos e você verá como localizar mais.

Entendendo uma declaração de API

A próxima linha é um exemplo de uma função API:

```
Private Declare Function GetUserNameLib "advapi32.dll" Alias _
  "GetUserNameA" (ByVal lpBuffer As String, nSize As Long)As Long
```

Há dois tipos de declarações de API: as funções, que retornam informações, e os procedimentos, que fazem algo para o sistema. As declarações são estruturadas de maneira semelhante.

Basicamente, o que a declaração anterior está dizendo é:

- É uma declaração `Private`, o que significa que só pode ser utilizada no módulo em que é declarada. Declare-a `Public` em um módulo-padrão se quiser compartilhá-la entre vários módulos.

ATENÇÃO

As declarações de API em módulos-padrão podem ser públicas ou privadas. As declarações de API em módulos de classe devem ser privadas.

- Esse procedimento será referido como `GetUserName` em seu programa. Esse é o nome da variável atribuído por você.
- A função em uso está localizada em advapi32.dll.
- O alias, `GetUserNameA`, é como a função é referida na DLL. Esse nome diferencia letras maiúsculas de minúsculas e não pode ser alterado; é específico à DLL. Freqüentemente, há duas versões de cada função API. Uma versão usa o conjunto de caracteres ANSI e tem aliases que acabam com a letra A.

A outra versão utiliza o conjunto de caracteres Unicode e tem aliases que terminam com a letra W. Ao especificar o alias, você está informando ao VBA que versão da função utilizar.

- Há dois parâmetros: `lpBuffer` e `nSize`. Esses são dois argumentos que a função DLL aceita.

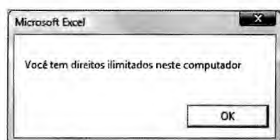
A desvantagem de utilizar APIs é que pode não haver nenhum erro quando o código estiver compilando ou executando. Uma chamada API configurada de modo incorreto pode fazer o computador cair ou travar. Por isso, é uma boa idéia salvar com freqüência.

Utilizando uma declaração API

Utilizar uma API não é diferente de chamar uma função ou um procedimento criado no VBA. O exemplo a seguir utiliza a declaração `GetUserName` em uma função para retornar o `UserName` no Excel:

```
Public Function UserName() As String
Dim sName As String * 256
Dim cChars As Long

cChars = 256
If GetUserName(sName, cChars) Then
    UserName = Left$(sName, cChars - 1)
End If
End Function
Sub ProgramRights()
Dim NameofUser As String

NameofUser = UserName

Select Case NameofUser
    Case Is = "Administrador"
        MsgBox "Você tem direitos ilimitados neste computador"
    Case Else
        MsgBox "Você tem direitos limitados neste computador"
End Select

End Sub
```

Execute a macro ProgramRights e você saberá se está conectado atualmente como administrador. O resultado mostrado na Figura 24.1 indica um administrador conectado.

Figura 24.1
A função `GetUserName` da API pode ser utilizada para obter o nome de login Windows de um usuário — o que é mais difícil de editar do que o nome de usuário do Excel.

Exemplos de API

As seções a seguir fornecem mais exemplos de declarações API úteis que podem ser utilizadas em seus programas Excel. Cada exemplo inicia com uma breve descrição do que o exemplo pode fazer, seguido da(s) declaração(ões) real(is) e de um exemplo de sua utilização.

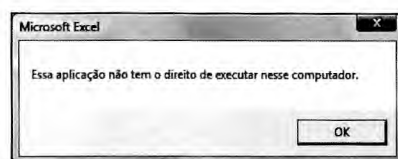
Recupere o nome do computador

Essa função API retorna o nome do computador. Esse é o nome do computador localizado em MyComputer, Identificação de Rede:

```
Private Declare Function GetComputerName Lib "kernel32" Alias"GetComputerNameA" (ByVal lpBuffer As String, ByRef
nSize As Long) As Long

Private Function ComputerName() As String

Dim stBuff As String * 255, lAPIResult As Long
Dim lBuffLen As Long

lBuffLen = 255
lAPIResult = GetComputerName(stBuff, lBuffLen)
```

```
If lBuffLen > 0 Then ComputerName = Left(stBuff, lBuffLen)

End Function

Sub ComputerCheck()
Dim CompName As String

CompName = ComputerName

If CompName <> "BillJelenPC" Then
    MsgBox _
    "Essa aplicação não tem direito de executar neste computador."
        ActiveWorkbook.Close SaveChanges:=False
End If

End Sub
```

A macro ComputerCheck utiliza uma chamada API para obter o nome do computador. Na Figura 24.2, o programa recusa executar em qualquer computador, exceto no computador do proprietário, cujo nome é codificado manualmente (*hard-coded*).

Figura 24.2
Utilize o nome do computador para verificar se um aplicativo tem o direito de executar no computador instalado.

Verifique se um arquivo Excel está aberto em uma rede

Você pode verificar se há um arquivo aberto no Excel tentando configurar a pasta de trabalho como um objeto. Se o objeto for `Nothing` (vazio), você saberá que o arquivo não está aberto. Mas e se você quiser saber se outra pessoa em uma rede está com o arquivo aberto? A próxima função API retorna essas informações:

```
Private Declare Function lOpen Lib "kernel32" Alias "_lopen"(ByVal lpPathName As String, ByVal iReadWrite As
Long) As Long

Private Declare Function lClose Lib "kernel32"Alias "_lclose" (ByVal hFile As Long) As Long

Private Const OF_SHARE_EXCLUSIVE = &H10
Private Function FileIsOpen(strFullPath_FileName As String) As Boolean
Dim hdlFile As Long
Dim lastErr As Long

hdlFile = -1

hdlFile = lOpen(strFullPath_FileName, OF_SHARE_EXCLUSIVE)

If hdlFile = -1 Then
    lastErr = Err.LastDllError
Else
    lClose (hdlFile)
End If

FileIsOpen = (hdlFile = -1) And (lastErr = 32)

End Function
Sub CheckFileOpen()

If FileIsOpen("C:\XYZ Corp.xlsx") Then
    MsgBox "O arquivo está aberto"
Else
    MsgBox "O arquivo não está aberto"
End If

End Sub
```

Chamar a função `FileIsOpen` com um caminho específico e o nome de arquivo como parâmetro o informará se alguém tem o arquivo aberto.

Recupere informações sobre a *resolução* de vídeo

A próxima função API recupera o tamanho de tela do computador:

```
Declare Function DisplaySize Lib "user32" Alias"GetSystemMetrics" (ByVal nIndex As Long) As Long

Public Const SM_CXSCREEN = 0
Public Const SM_CYSCREEN = 1

Function VideoRes() As String
Dim vidWidth
Dim vidHeight

vidWidth = DisplaySize(SM_CXSCREEN)
vidHeight = DisplaySize(SM_CYSCREEN)

Select Case (vidWidth * vidHeight)
    Case 307200
        VideoRes = "640 x 480"
    Case 480000
        VideoRes = "800 x 600"
    Case 786432
        VideoRes = "1024 x 768"
    Case Else
        VideoRes = "Algo mais"
End Select

End Function

Sub CheckDisplayRes()
Dim VideoInfo As String
Dim Msg1 As String, Msg2 As String, Msg3 As String

VideoInfo = VideoRes

Msg1 = "A resolução atual está configurada em " & VideoInfo & Chr(10)
Msg2 = "A melhor resolução para essa aplicação é 1024 x 768" & Chr(10)
Msg3 = "Ajuste a resolução"

Select Case VideoInfo
    Case Is = "640 x 480"
        MsgBox Msg1 & Msg2 & Msg3
    Case Is = "800 x 600"
        MsgBox Msg1 & Msg2
    Case Is = "1024 x 768"
        MsgBox Msg1
    Case Else
        MsgBox Msg2 & Msg3
End Select

End Sub
```

A macro CheckDisplayRes adverte o cliente de que a configuração de vídeo não é ideal para o aplicativo. Verifique a resolução de vídeo e solicite que o usuário altere a resolução a fim de aproveitar mais o aplicativo.

Personalize a caixa de diálogo Sobre

Se consultar a Ajuda e Suporte do Windows no Windows Explorer, você obterá uma pequena e ótima caixa de diálogo Sobre com informações sobre o Windows Explorer e alguns detalhes de sistema. Com o código a seguir, você poderá abrir essa janela no próprio programa e personalizar alguns itens, como mostrado na Figura 24.3.

Figura 24.3
Você pode personalizar a caixa de diálogo Sobre utilizada pelo Windows para o seu próprio programa.

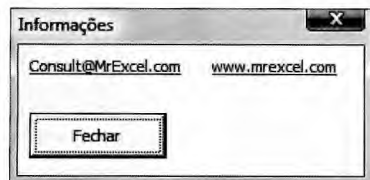
```
Declare Function ShellAbout Lib "shell32.dll" Alias "ShellAboutA
    (ByVal hwnd As Long, ByVal szApp As String, ByVal szOtherStuff As String,ByVal hIcon As Long) As Long
Declare Function GetActiveWindow Lib "user32" () As Long

Sub AboutMrExcel()
    Dim hwnd As Integer
    On Error Resume Next
    hwnd = GetActiveWindow()
    ShellAbout hwnd, Nm, vbCrLf + Chr(169) + "" & " MrExcel.com Consulting"+ vbCrLf, 0
    On Error GoTo 0
End Sub
```

Desative o X para fechar um userform

No canto superior direito de um userform, há um botão X que pode ser utilizado para desativar o aplicativo. As declarações API seguintes trabalham em conjunto para desativar esse X, forçando o usuário a utilizar o botão Fechar. Quando o formulário é inicializado, o botão é desativado. Depois que o formulário é fechado, o botão X é redefinido como normal:

```
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA"(ByVal lpClassName As String, ByVal
lpWindowName As String) As Long
Private Declare Function GetSystemMenu Lib "user32" (ByVal hWnd As Long,ByVal bRevert As Long) As Long
Private Declare Function DeleteMenu Lib "user32"(ByVal hMenu As Long, ByVal nPosition As Long,ByVal wFlags As
Long) As Long
Private Const SC_CLOSE As Long = &HF060

Private Sub UserForm_Initialize()
Dim hWndForm As Long
Dim hMenu As Long

hWndForm = FindWindow("ThunderDFrame", Me.Caption)  'XL2000
hMenu = GetSystemMenu(hWndForm, 0)
DeleteMenu hMenu, SC_CLOSE, 0&

End Sub
```

A macro DeleteMenu no procedimento `UserForm_Initialize` faz com que o X no canto do userform seja desativado, como mostrado na Figura 24.4. Isso força o cliente a utilizar o botão Fechar programado.

Figura 24.4
Desative o botão X em um userform, forçando os usuários a utilizar o botão Fechar para desativar o formulário adequadamente, de modo que nenhum código anexado ao botão Fechar seja pulado.

Cronômetro contínuo

Você pode utilizar a função NOW para obter a hora, mas e se você precisasse de um cronômetro contínuo? Um cronômetro que exibisse a hora exata a cada segundo? As declarações API a seguir trabalham em conjunto para fornecer essa funcionalidade. O cronômetro é colocado na célula A1 de Sheet1:

```
Public Declare Function SetTimer Lib "user32"(ByVal hWnd As Long, ByVal nIDEvent As Long,ByVal uElapse As Long, _
    ByVal lpTimerFunc As Long) As Long
Public Declare Function KillTimer Lib "user32"(ByVal hWnd As Long, ByVal nIDEvent As Long) As Long
Public Declare Function FindWindow Lib "user32"Alias "FindWindowA" (ByVal lpClassName As String,ByVal
lpWindowName As String) As Long

Private lngTimerID As Long
Public datStartingTime As Date

Public Sub StartTimer()
lngTimerID = SetTimer(0, 1, 10, AddressOf RunTimer)
End Sub

Public Sub StopTimer()
Dim lRet As Long
lRet = KillTimer(0, lngTimerID)
End Sub

Private Sub RunTimer(ByVal hWnd As Long,ByVal uint1 As Long, ByVal nEventId As Long,ByVal dwParam As Long)
On Error Resume Next
Sheet1.Range("A1").Value = Now - datStartingTime
End Sub
```

Execute a macro StartTimer para obter a data e hora atuais constantemente atualizadas na célula A1.

Reproduzindo sons

Alguma vez você já quis tocar um som para alertar ou parabenizar os usuários? Você poderia adicionar um objeto de som a uma planilha e chamar isso, mas é muito mais fácil utilizar apenas a declaração API seguinte e especificar o caminho adequado para um arquivo de som:

```
Public Declare Function PlayWavSound Lib "winmm.dll"Alias "sndPlaySoundA" (ByVal LpszSoundName As String,ByVal
uFlags As Long) As Long

Public Sub PlaySound()
Dim SoundName As String
SoundName = "C:\WinNT\Media\Chimes.wav"
PlayWavSound SoundName, 0
End Sub
```

Recuperando um caminho de arquivo

A API a seguir permite a criação de um navegador de arquivo personalizado. O exemplo de programa que utiliza a API personaliza a chamada de função para criar um navegador para uma necessidade específica — nesse caso, retornar o caminho de arquivo de um arquivo selecionado pelo usuário:

```
Type tagOPENFILENAME
    lStructSize As Long
    hwndOwner As Long
    hInstance As Long
    strFilter As String
    strCustomFilter As String
    nMaxCustFilter As Long
    nFilterIndex As Long
    strFile As String
    nMaxFile As Long
    strFileTitle As String
    nMaxFileTitle As Long
    strInitialDir As String
    strTitle As String
    Flags As Long
    nFileOffset As Integer
    nFileExtension As Integer
```

```
    strDefExt As String
    lCustData As Long
    lpfnHook As Long
    lpTemplateName As String
End Type

Declare Function aht_apiGetOpenFileName Lib "comdlg32.dll"Alias "GetOpenFileNameA" (OFN As tagOPENFILENAME) As _
    Boolean
Declare Function aht_apiGetSaveFileName Lib "comdlg32.dll"Alias "GetSaveFileNameA" (OFN As tagOPENFILENAME) As _
    Boolean
Declare Function CommDlgExtendedError Lib "comdlg32.dll" () As Long

Global Const ahtOFN_READONLY = &H1
Global Const ahtOFN_OVERWRITEPROMPT = &H2
Global Const ahtOFN_HIDEREADONLY = &H4
Global Const ahtOFN_NOCHANGEDIR = &H8
Global Const ahtOFN_SHOWHELP = &H10
Global Const ahtOFN_NOVALIDATE = &H100
Global Const ahtOFN_ALLOWMULTISELECT = &H200
Global Const ahtOFN_EXTENSIONDIFFERENT = &H400
Global Const ahtOFN_PATHMUSTEXIST = &H800
Global Const ahtOFN_FILEMUSTEXIST = &H1000
Global Const ahtOFN_CREATEPROMPT = &H2000
Global Const ahtOFN_SHAREAWARE = &H4000
Global Const ahtOFN_NOREADONLYRETURN = &H8000
Global Const ahtOFN_NOTESTFILECREATE = &H10000
Global Const ahtOFN_NONETWORKBUTTON = &H20000
Global Const ahtOFN_NOLONGNAMES = &H40000
Global Const ahtOFN_EXPLORER = &H80000
Global Const ahtOFN_NODEREFERENCELINKS = &H100000
Global Const ahtOFN_LONGNAMES = &H200000

Function ahtCommonFileOpenSave(Optional ByRef Flags As Variant,Optional ByVal InitialDir As Variant,Optional _
    ByVal Filter As Variant,Optional ByVal FilterIndex As Variant,Optional ByVal DefaultExt As Variant,Optional _
    ByVal FileName As Variant,Optional ByVal DialogTitle As Variant,Optional ByVal hwnd As Variant,Optional _
    ByVal OpenFile As Variant) As Variant

'Esse é o ponto de entrada que você utilizará para chamar a caixa de diálogo Abrir/Salvar Como.
'Os parâmetros são listados abaixo e todos são opcionais.
'
'In:
'Sinalizadores: uma ou mais das constantes ahtOFN_* ou OR'd juntas.
'InitialDir: o diretório ao qual examinar primeiro
'Filtro: um conjunto de filtros de arquivo (Utilize AddFilterItem para configurar Filtros)
'Índice de filtro: inteiro baseado em 1 que indica qual filtro utilizar, por padrão (1 se não especificado)
'Extensão padrão: A extensão a ser usada se o usuário não inserir uma. Útil apenas para salvar o arquivo.
'Nome do arquivo: Valor padrão para a caixa de texto do nome de arquivo.
'Título do diálogo: Título para a caixa de diálogo.
'hWnd: handle da janela pai
'OpenFile: Boolean(True=Open File/False=Save As)
' Saída:
'Valor de retorno: Nulo ou o nome de arquivo selecionado

Dim OFN As tagOPENFILENAME
Dim strFileName As String
Dim strFileTitle As String
Dim fResult As Boolean

'Fornece um título de legenda para a caixa de diálogo.
If IsMissing(InitialDir) Then InitialDir = CurDir
If IsMissing(Filter) Then Filter = ""
If IsMissing(FilterIndex) Then FilterIndex = 1
If IsMissing(Flags) Then Flags = 0&
If IsMissing(DefaultExt) Then DefaultExt = ""
If IsMissing(FileName) Then FileName = ""
If IsMissing(DialogTitle) Then DialogTitle = ""
If IsMissing(OpenFile) Then OpenFile = True

'Aloca espaço de string para as strings retornadas.
strFileName = Left(FileName & String(256, 0), 256)
strFileTitle = String(256, 0)
```

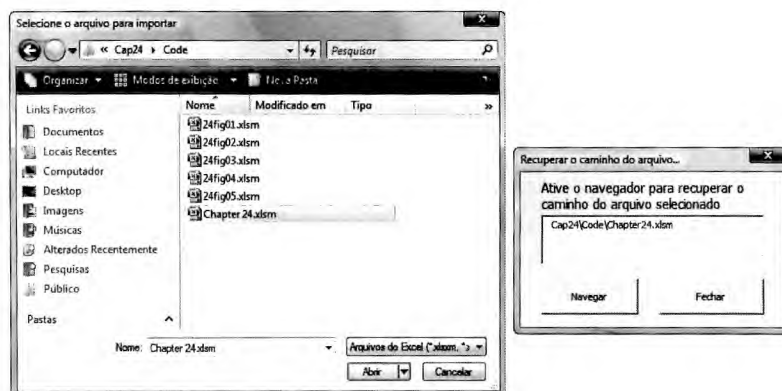
```
'Configura a estrutura de dados antes de você chamar a função
With OFN
    .lStructSize = Len(OFN)
    .strFilter = Filter
    .nFilterIndex = FilterIndex
    .strFile = strFileName
    .nMaxFile = Len(strFileName)
    .strFileTitle = strFileTitle
    .nMaxFileTitle = Len(strFileTitle)
    .strTitle = DialogTitle
    .Flags = Flags
    .strDefExt = DefaultExt
    .strInitialDir = InitialDir
    .hInstance = 0
    .lpfnHook = 0
    .strCustomFilter = String(255, 0)
    .nMaxCustFilter = 255
End With

'Isso passa a estrutura de dados desejada para a
'API do Windows, que por sua vez exibirá
'a caixa de diálogo Abrir/Salvar Como.
If OpenFile Then
    fResult = aht_apiGetOpenFileName(OFN)
Else
    fResult = aht_apiGetSaveFileName(OFN)
End If

'A chamada de função preenchida no membro strFileTitle
'da estrutura. Você tem de escrever código especial
'para recuperar isso se estiver interessado.
If fResult Then
'Você poderia tomar o cuidado de verificar o membro Flags da
'estrutura para obter informações sobre o arquivo escolhido.
'Nesse exemplo, se você se incomodou em passar um
'Valor de flags, a serem preenchidos com a saída
'Valor de flags.
    If Not IsMissing(Flags) Then Flags = OFN.Flags
        ahtCommonFileOpenSave = TrimNull(OFN.strFile)
    Else
        ahtCommonFileOpenSave = vbNullString
    End If

End Function

Function ahtAddFilterItem(strFilter As String,strDescription As String, Optional varItem As Variant) As String
'Fornece um novo pedaço para o filtro de arquivo.
'Isto é, pega o antigo valor, coloca-o sobre a descrição,
'(como "Databases"), um caractere nulo, o esqueleto
'(Como "*.mdb;*.mda") e um caractere nulo final.

If IsMissing(varItem) Then varItem = "*.*"
ahtAddFilterItem = strFilter & strDescription &vbNullChar & varItem & vbNullChar

End Function

Private Function TrimNull(ByVal strItem As String) As String
Dim intPos As Integer

intPos = InStr(strItem, vbNullChar)

If intPos > 0 Then
    TrimNull = Left(strItem, intPos - 1)
Else
    TrimNull = strItem
End If

End Function
```

Esse é o programa real criado para utilizar essas informações:

```
Function GetFileName(strPath As String)
Dim strFilter As String
Dim lngFlags As Long

strFilter = ahtAddFilterItem(strFilter, "Arquivos do Excel (*.xls)")
GetFileName = ahtCommonFileOpenSave(InitialDir:=strPath,Filter:=strFilter, FilterIndex:=3, Flags:=lngFlags,Dialog
Title:="Selecione o arquivo para importar")

End Function
```

Agora crie o userform. O código seguinte é anexado ao botão Procurar, como mostrado na Figura 24.5. Observe que a função especifica o diretório inicial:

```
Private Sub cmdBrowse_Click()

txtFile = GetFileName("c:\")

End Sub
```

Figura 24.5
Crie uma janela de navegação personalizada para retornar o caminho de arquivo de um arquivo selecionado pelo usuário. Isso pode ser utilizado para assegurar que o usuário não selecione o arquivo errado para importar.

Localizando mais declarações API

Há muito mais declarações API do que aquelas que apresentamos aqui — tocamos apenas na ponta do iceberg da grande quantidade de procedimentos e funções disponíveis. A Microsoft dispõe de muitas ferramentas para ajudá-lo a criar suas próprias APIs (pesquise Platform SDK), mas também há muitos programadores que desenvolveram declarações para compartilhar, como Ivan F. Moala em http://xcelfiles.homestead.com/APIIndex.html. Ele criou um site Web repleto não apenas de exemplos, mas também de instruções.

Próximos passos

No Capítulo 25, "Tratando erros", você aprenderá sobre tratamento de erros. Em um mundo perfeito, você pode passar seus aplicativos para um colaborador durante as férias sem precisar se preocupar com um erro não-tratado que pode aparecer enquanto você estiver na praia. O Capítulo 25 discute como tratar erros óbvios e erros não tão óbvios.

Tratando erros

25

NESTE CAPÍTULO

Erros são inevitáveis. Você pode testar e retestar o código, mas depois de um relatório ser colocado na produção diária e utilizado por centenas de dias, algo inesperado por fim acontece. Seu objetivo deve ser tentar impedir erros obscuros durante a codificação. Esteja sempre pensando em coisas inesperadas que poderão acontecer algum dia e que farão com que o código deixe de funcionar.

O que acontece quando ocorre um erro

Se o VBA encontrar um erro e não houver nenhum código de verificação de erro instalado, o programa irá parar e a mensagem de erro 'Continuar, Fim, Depurar, Ajuda' será apresentada para você (ou seu cliente), como mostrado na Figura 25.1.

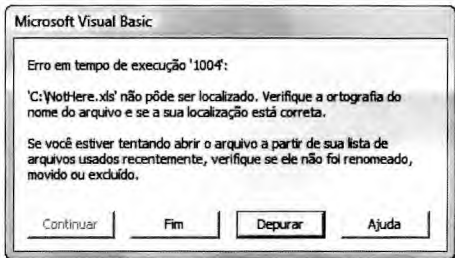
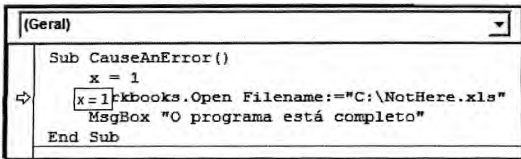
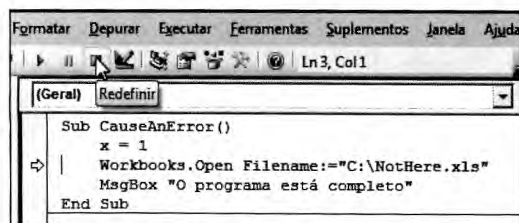
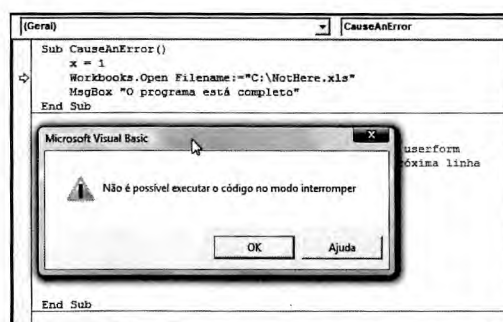
Figura 25.1
Um erro não tratado em um módulo desprotegido apresenta uma opção Fim ou Depurar.

Quando a opção Fim ou Depurar for apresentada, você deve clicar em Depurar. O Editor do VB realçará em amarelo a linha que causou o erro. Você pode posicionar o cursor sobre qualquer variável para ver o seu valor atual. Isso fornece muitas informações sobre o que poderia ter causado o erro (veja Figura 25.2).

Figura 25.2
Depois de clicar em Depurar, a macro entra no modo de interrupção. Você pode posicionar o cursor sobre uma variável; depois de alguns segundos, o valor atual da variável é mostrado.

O Excel é notório por retornar erros que não são muito significativos. Por exemplo, várias situações podem causar um erro 1004. Ser capaz de visualizar a linha problemática destacada em amarelo e examinar o valor atual de qualquer variável ajudam a descobrir a causa real de um erro.

Depois de examinar a linha com o erro, clique no botão Redefinir para interromper a execução da macro. O botão Redefinir é o botão quadrado sob o item Executar no menu principal, como mostrado na Figura 25.3.

Figura 25.3
O botão Redefinir é parecido com o botão Parar no conjunto de três botões que lembra um painel de controle de um videocassete.

ATENÇÃO

Se não conseguir clicar em Redefinir para interromper a macro e, então, tentar executar outra macro, você receberá uma mensagem de erro irritante, conforme mostrado na Figura 25.4. A mensagem é irritante porque você inicia no Excel mas, quando essa janela de mensagem é exibida, a tela muda automaticamente para exibir o Editor do VB. Porém, logo que você clica em OK, a tela retorna ao Excel em vez de ficar no Editor do VB. Pela freqüência com que essa mensagem aparece, seria muito mais conveniente se você pudesse permanecer no VB depois de clicar em OK.

Figura 25.4
Essa mensagem aparecerá se você se esquecer de clicar em Redefinir para terminar uma sessão de depuração e, então, tentar executar outra macro.

Erro de depuração dentro de código de userform é enganoso

A linha que foi destacada como erro depois de você clicar em Depurar pode ser enganosa em uma situação. Suponha que você chame uma macro. Essa macro então exibe um userform. Em algum lugar no código do userform ocorre um erro. Quando você clica em Depurar, o Excel destaca a linha na macro original que exibiu o userform em vez de mostrar o problema dentro do código do userform. Siga estes passos para localizar o erro real.

1. Depois que uma caixa de mensagem de erro é mostrada, como na Figura 25.5, clique no botão Depurar.

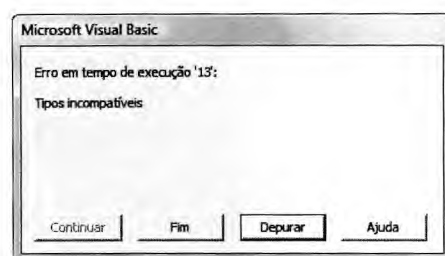
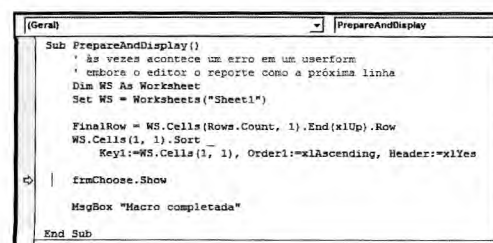
Figura 25.5
Escolha Depurar em resposta a esse erro 13.

Você verá que o erro deve ter ocorrido em uma linha que exibe um userform, como mostrado na Figura 25.6. Como você leu este capítulo, sabe que essa não é a linha com erro.

Figura 25.6
`frmChoose.Show` é indicada como a linha com erro.

2. Pressione F8 para executar o método `Show`. Em vez de obter um erro, você será levado para o procedimento `Userform_Initialize`.

3. Mantenha a tecla F8 pressionada até obter a mensagem de erro outra vez. Fique alerta. Logo que você encontrar o erro, a caixa de mensagem de erro será exibida. Clique em Depurar para retornar à linha `userform.Show`. É particularmente difícil seguir o código quando o erro ocorre no outro lado de um loop longo, como mostrado na Figura 25.7.

Imagine-se tentando percorrer o código da Figura 25.7. Você pressiona F8 cuidadosamente cinco vezes sem nenhum problema pela primeira passagem do loop. Como o problema pode estar nas futuras iterações pelo loop, você continua a pressionar F8. Se houver 25 itens para adicionar à caixa de listagem, mais 48 pressionamentos da tecla F8 são necessários para percorrer o loop com segurança.

Toda vez antes de pressionar F8, você deve observar mentalmente que está prestes a executar alguma linha específica.

Figura 25.7
Com 25 itens para ser adicionados à caixa de listagem, você deve pressionar F8 51 vezes para passar por esse loop de três linhas.

```
Private Sub CommandButton1_Click()
    Unload Me
End Sub

Private Sub UserForm_Initialize()
    Dim WS As Worksheet
    Set WS = Worksheets("Sheet1")

    FinalRow = WS.Cells(Rows.Count, 1).End(xlUp).Row
    For i = 2 To FinalRow
        Me.ListBox1.AddItem WS.Cells(i, 1)
    Next i

    ' A próxima linha é realmente a linha que causa erro
⇨   Me.ListBox1(0).Selected = True

End Sub
```

No ponto mostrado na Figura 25.7, o próximo pressionamento da tecla F8 exibe o erro e o retorna à linha `frmChoose.Show` em Module1.

Essa é uma situação irritante. Quando clicar em Depurar e vir que a linha com erro é uma linha que exibe um userform, comece a pressionar a tecla F8 para entrar no código do userform até chegar ao erro. Sempre fico muito entediado em ter de pressionar F8 um milhão de vezes e esqueço de prestar atenção na linha que causou o erro. Logo que o erro acontece, sou lançado de volta para a mensagem de depuração, que me retorna para a linha `frmChoose.Show` do código. Nesse ponto, é preciso começar a pressionar F8 novamente. Se você se lembrar da área geral em que o erro de depuração ocorreu, clique no cursor do mouse em uma linha logo antes dessa seção e pressione Ctrl+F8 para executar a macro até o cursor.

Tratamento de erro básico com a sintaxe `On Error GoTo`

A opção de tratamento de erro básica serve para dizer ao VBA que, em caso de erro, você quer que o código seja desviado para uma área específica da macro. Nessa área, você pode ter código especial que alerta usuários de problemas e permite que eles reajam.

Um cenário típico é adicionar a rotina de tratamento de erro no fim da macro. Para configurar uma rotina de tratamento de erro, siga estes passos:

1. Depois da última linha de código da macro, insira a linha de código `Exit Sub`. Isso assegura que a execução da macro não continua na rotina de tratamento de erro.

2. Depois da linha `Exit Sub`, adicione um rótulo. Rótulo é um nome seguido por um dois-pontos. Por exemplo, você poderia criar um rótulo chamado MyErrorHandler:.

3. Escreva o código para tratar o erro. Se quiser retornar o controle da macro à linha depois daquela que causou o erro, utilize a instrução `Resume Next`.

Em sua macro, antes da linha que provavelmente causou o erro, adicione uma linha exibindo On Error GoTo MyErrorHandler. Observe que, nessa linha, você não inclui o dois-pontos depois do nome de rótulo.

Imediatamente depois da linha de código que você suspeita ser a causa do erro, adicione código para desativar a rotina de tratamento de erro especial. Isso é muito não-intuitivo e tende a confundir as pessoas. O código para cancelar qualquer tratamento de erro especial é `On Error Goto 0`. Não há rótulo nomeado 0. Essa é uma linha fictícia que instrui o Excel a voltar para o estado normal de exibição da mensagem de erro Fim/Depurar quando um erro é encontrado. Você vê por que é importante cancelar o tratamento de erro? No próximo código, escrevemos uma rotina de tratamento de erro especial para tratar a ação necessária se o arquivo tiver sido movido ou estiver ausente. Definitivamente, não queremos essa rotina de tratamento de erro invocada por outro erro mais tarde na macro, como uma divisão por zero:

```
Sub HandleAnError()
    Dim MyFile as Variant
    ' Configurando uma rotina de tratamento de erro especial
    On Error GoTo FileNotThere
```

```
    Workbooks.Open Filename:="C:\NotHere.xls"
    ' Se chegamos aqui, cancelamos a rotina de tratamento de erro especial
    On Error GoTo 0
    MsgBox "O programa está completo"

    ' A macro está concluída. Utilize Exit Sub, caso contrário a execução da macro
    ' IRÁ continuar na rotina de tratamento de erro
    Exit Sub

    ' Configura um nome para a rotina de tratamento de erro
FileNotThere:
    MyPrompt = "Houve um erro ao abrir o arquivo. É possível que o "
    MyPrompt = MyPrompt & " arquivo tenha sido transferido. Clique em OK para procurar o "
    MyPrompt = MyPrompt & "arquivo, ou clique em Cancelar para finalizar o programa"
    Ans = MsgBox(Prompt:=MyPrompt, VbMsgBoxStyle:=vbOKCancel)
    If Ans = vbCancel Then Exit Sub

    ' O cliente clicou em OK. Deixe-o procurar o arquivo
    MyFile = Application.GetOpenFilename
    If MyFile = False Then Exit Sub

    ' E se o arquivo 2 estiver corrompido? Não queremos lançar recursivamente
    ' o cliente de volta nessa rotina de tratamento de erro. Simplesmente pára o programa
    On Error GoTo 0
    Workbooks.Open MyFile
    ' Se chegamos aqui, então retornamos a execução da macro de volta à seção original
    ' da macro, para uma linha depois daquela que causou o erro.
    Resume Next

End Sub
```

DICA É possível ter mais de uma rotina de tratamento de erro no fim de uma macro. Certifique-se de que cada rotina de tratamento de erro acaba com `Resume Next` ou `Exit Sub` para que a execução da macro não entre acidentalmente na próxima rotina de tratamento de erro.

Rotinas de tratamento de erro genéricas

Alguns desenvolvedores gostam de direcionar qualquer erro para uma rotina de tratamento de erro genérica. Eles utilizam o objeto `Err`. Esse objeto tem propriedades para número e descrição de erro. Você pode oferecer essas informações para o cliente e impedir que eles recebam uma mensagem de depuração:

```
    On Error GoTo HandleAny
    Sheets(9).Select

    Exit Sub

HandleAny:
    Msg = "Nós encontramos " & Err.Number & " - " & Err.Description
    MsgBox Msg
    Exit Sub
```

Tratando erros ao escolher ignorá-los

Alguns erros podem simplesmente ser ignorados. Suponha, por exemplo, que você vá utilizar a macro HTML Creator do Capítulo 16, "Lendo e gravando na Web". Seu código apaga qualquer arquivo index.html existente em uma pasta antes de gravar o próximo arquivo.

A instrução `Kill (FileName)` retorna um erro se `FileName` não existir. Isso é algo com que você deve se preocupar? Você está tentando excluir o arquivo. Quem é responsável se alguém já o excluiu antes de a macro ser executada? Nesse caso, você deve informar o Excel para simplesmente pular a linha com problema e retomar a execução da macro com a próxima linha. O código para fazer isso é `On Error Resume Next`:

```
Sub WriteHTML()
    MyFile = "C:\Index.html"
    On Error Resume Next
    Kill (MyFile)
```

```
        On Error Goto 0
        Open MyFile for Output as #1
        ' etc.
    End Sub
```

Tome cuidado com a sintaxe `On Error Resume Next`, que pode ser utilizada seletivamente nas situações em que você sabe que o erro pode ser ignorado. Você deve retornar imediatamente a verificação de erros ao normal, depois da linha que pode causar um erro, com a sintaxe `On Error GoTo 0`.

Se tentar fazer `On Error Resume Next` pular um erro que não pode ser pulado, a macro imediatamente sairá da macro atual. Se você tiver uma situação em que a MacroA chama a MacroB e a MacroB encontrar um erro que não pode ser pulado, o programa sairá da MacroB e continuará com a próxima linha na MacroA. Poucas vezes isso é uma coisa boa.

Estudo de caso

Problemas de configuração de página podem ser com freqüência ignorados

Grave uma macro e realize uma configuração de página. Mesmo que você altere apenas um item na caixa de diálogo Configurar Página, o programa de gravação de macros gravará diversas configurações para você. Essas configurações são, notoriamente, diferentes de impressora para impressora. Por exemplo, se você gravar PageSetup em um sistema com impressora colorida, poderá gravada uma configuração para `.BlackAndWhite = True`. Essa configuração falhará em outro sistema em que a impressora não oferecer escolha. Sua impressora pode oferecer uma configuração `.PrintQuality = 600`. Se a impressora do cliente oferecer uma resolução de apenas 300, esse código falhará. Coloque a PageSetup inteira entre `On Error Resume Next` para assegurar que se qualquer configuração sem importância falhar, isso não cause um erro de tempo de execução:

```
On Error Resume Next
With ActiveSheet.PageSetup
    .PrintTitleRows = ""
    .PrintTitleColumns = ""
End With
ActiveSheet.PageSetup.PrintArea = "$A$1:$L$27"
With ActiveSheet.PageSetup
    .LeftHeader = ""
    .CenterHeader = ""
    .RightHeader = ""
    .LeftFooter = ""
    .CenterFooter = ""
    .RightFooter = ""
    .LeftMargin = Application.InchesToPoints(0.25)
    .RightMargin = Application.InchesToPoints(0.25)
    .TopMargin = Application.InchesToPoints(0.75)
    .BottomMargin = Application.InchesToPoints(0.5)
    .HeaderMargin = Application.InchesToPoints(0.5)
    .FooterMargin = Application.InchesToPoints(0.5)
    .PrintHeadings = False
    .PrintGridlines = False
    .PrintComments = xlPrintNoComments
    .PrintQuality = 300
    .CenterHorizontally = False
    .CenterVertically = False
    .Orientation = xlLandscape
    .Draft = False
    .PaperSize = xlPaperLetter
    .FirstPageNumber = xlAutomatic
    .Order = xlDownThenOver
    .BlackAndWhite = False
    .Zoom = False
    .FitToPagesWide = 1
    .FitToPagesTall = False
    .PrintErrors = xlPrintErrorsDisplayed
End With
On Error GoTo 0
```

Removendo avisos do Excel

Algumas mensagens são exibidas mesmo que você tenha configurado o Excel para ignorar erros. Tente excluir uma planilha utilizando código e ainda receberá a mensagem "Data may exist in the sheet(s) selected for deletion" [Pode haver dados na(s) planilha(s) selecionados para exclusão). Para excluir de modo permanente os dados, clique em Excluir. Isso é irritante. Não queremos que nossos clientes tenham de responder a esse aviso. Acontece que isso não é um erro, mas um alerta. Para remover todos os alertas e forçar o Excel tomar a ação-padrão, utilize `Application.DisplayAlerts = False`:

```
Sub DeleteSheet()
    Application.DisplayAlerts = False
    Worksheets("Sheet2").Delete
    Application.DisplayAlerts = True
End Sub
```

Encontrando erros de propósito

Como os programadores odeiam erros, esse conceito pode parecer absurdo, mas os erros nem sempre são ruins. Às vezes, é mais rápido simplesmente encontrar um erro.

Digamos que você queira descobrir se a pasta de trabalho ativa contém uma planilha chamada Dados. Para localizar essa planilha sem causar um erro, você poderia codificar isto:

```
DataFound = False
For each ws in ActiveWorkbook.Worksheets
    If ws.Name = "Dados" then
        DataFound = True
        Exit For
    End if
Next ws
If not DataFound then Sheets.Add.Name = "Dados"
```

Isso requer oito linhas de código. Se sua pasta de trabalho tivesse 128 planilhas, o programa faria loop 128 vezes antes de descobrir que a planilha Dados não existe.

A alternativa é simplesmente tentar referenciar a planilha Dados. Se a verificação de erros estiver definida para retomar em seguida, o código executará e ao objeto `Err` será atribuído um número diferente de zero:

```
On Error Resume Next
X = Worksheets("Dados").Name
If not Err.Number = 0 then Sheets.Add.Name = "Dados"
On Error GoTo 0
```

Esse código executa muito mais rapidamente. Em geral, tenho medo dos erros; nesse caso, porém e, em muitos outros casos, eles são perfeitamente aceitáveis.

Treine seus clientes

Você pode desenvolver código para um cliente do outro lado do mundo ou para um assistente administrativo para que ele possa executar o código enquanto você estiver de férias. Em qualquer caso, pode ser que o cliente o procure para que você tente depurar o código por telefone.

É importante treinar os clientes sobre a diferença entre um erro e uma MsgBox simples. Uma MsgBox é uma mensagem planejada que aparece inesperadamente com um bip. Ensine os usuários que as mensagens de erro são ruins, mas que nem tudo que aparece na tela é uma mensagem de erro. Tive uma cliente que queixava-se sempre com seu patrão de que recebia um erro do meu programa. Na realidade, ela estava recebendo uma MsgBox informacional. Tanto os erros de depuração quanto as mensagens Msgbox emitem um som para o usuário.

Quando os clientes obtiverem erros de depuração, peça-lhes para que entrem em contato com você enquanto a mensagem de Depuração ainda estiver na tela. Assim, é possível obter o número e a descrição do erro e pedir que eles cliquem em Depurar e informem o nome do módulo, o nome do procedimento e a linha em amarelo. Munido dessas informações, em geral você descobre o que está acontecendo. Mas, sem elas, é improvável que consiga descobrir o problema. Receber uma ligação de um cliente dizendo que há um erro 1004 não ajuda muito — 1004 é um erro que abrange tudo e que pode significar inúmeras coisas.

Erros durante o desenvolvimento *versus* erros meses mais tarde

Quando acaba de escrever o código e o está executando pela primeira vez, você espera erros. De fato, você pode optar por percorrer o código linha por linha para observar o progresso dele na primeira vez.

A história é outra quando um programa que estava em produção diariamente pára de repente de funcionar com um erro. Isso parece confuso; o código esteve funcionando durante meses. Por que ele parou repentinamente de funcionar hoje?

É fácil culpar o cliente por isso; mas você acaba descobrindo que a falha foi realmente dos desenvolvedores, que não consideraram as possibilidades.

As seções a seguir descrevem dois problemas comuns que podem atacar um aplicativo meses mais tarde.

Erro em tempo de execução '9': subscrito fora do intervalo

Você configurou um aplicativo para o cliente. Forneceu uma planilha Menu em que armazena algumas configurações. Um dia, o cliente informa que recebeu a mensagem de erro mostrada na Figura 25.8.

Seu código esperava que existisse uma planilha chamada Menu, mas, por alguma razão, o cliente excluiu ou renomeou acidentalmente a planilha. Assim que tentou selecioná-la, ele recebeu uma mensagem de erro:

```
Sub GetSettings()
    ThisWorkbook.Worksheets("Menu").Select
    x = Range("A1").Value
End Sub
```

Figura 25.8
Em geral, o erro em tempo de execução '9' é causado quando esperamos a presença de uma planilha que foi excluída ou renomeada pelo cliente.

Essa é uma situação clássica, que ocorre porque você não consegue acreditar que o cliente faria algo tão louco. Depois de deparar-se com isso algumas vezes, talvez você não meça esforços para impedir que um erro de depuração não tratado venha a ocorrer:

```
Sub GetSettings()
    On Error Resume Next
    x = ThisWorkbook.Worksheets("Menu").Name
    If Not Err.Number = 0 Then
        MsgBox "Era esperado encontrar uma planilha Menu, mas esta está ausente"
        Exit Sub
    End If
    On Error GoTo 0

    ThisWorkbook.Worksheets("Menu").Select
    x = Range("A1").Value
End Sub
```

Erro em tempo de execução '1004': o método `Range` do objeto `Global` falhou

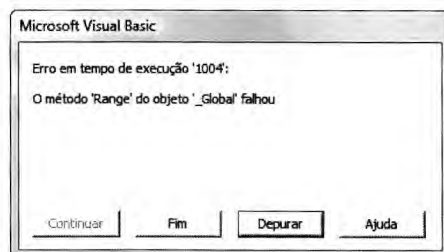
Você tem código que importa um arquivo de texto diariamente. Espera que o arquivo de texto termine com uma linha Total. Depois de importar o texto, você quer converter todas as linhas de detalhe em itálico.

O próximo código funciona bem durante meses:

```
Sub SetReportInItalics()
    TotalRow = Cells(Rows.Count,1).End(xlUp).Row
    FinalRow = TotalRow - 1
    Range("A1:A" & FinalRow).Font.Italic = True
End Sub
```

Então, um dia, o cliente liga informando que recebeu a mensagem de erro mostrada na Figura 25.9.

Ao examinar o código, você descobre que algo esquisito saiu errado quando o arquivo de texto foi transferido via FTP para o cliente naquele dia. O arquivo de texto terminou como um arquivo vazio. Como a planilha estava vazia, `TotalRow` foi determinada como a Linha 1. Quando se assume que a última linha de detalhe é `TotalRow 1`, o código é definido para tentar formatar a Linha 0, que claramente não existe.

Figura 25.9
O erro em tempo de execução '1004' pode ser causado por inúmeras coisas.

Depois desse episódio, você se verá escrevendo código considerando antecipadamente essa situação:

```
Sub SetReportInItalics()
    TotalRow = Cells(Rows.Count,1).End(xlUp).Row
    FinalRow = TotalRow - 1
    If FinalRow > 0 Then
        Range("A1:A" & FinalRow).Font.Italic = True
    Else
        MsgBox "Parece que o arquivo está cheio atualmente. Verifique o processo de FTP"
    End If
End Sub
```

Os aspectos desfavoráveis de proteger o código

É possível bloquear um projeto VBA para que ele não seja visualizado. Não recomendo isso de modo algum. Quando o código é protegido e um erro é encontrado, seu usuário recebe uma mensagem de erro, mas sem nenhuma oportunidade de depuração. O botão Depurar está presente, mas está desativado. Isso é inútil para ajudar a descobrir o problema.

Além disso, o esquema de proteção do Excel VBA é, infelizmente, muito fácil de quebrar. Programadores na Estônia oferecem software de US$ 40 que permite desbloquear qualquer projeto. Então, entenda que o código Office VBA não é seguro e supere-o.

Estudo de caso

Quebra de senha

Os esquemas de hacking de senhas eram muito fáceis no Excel 97 e 2000. O software de quebra de senhas podia localizar imediatamente a senha real no projeto VBA e informá-la para o usuário do software.

Então, no Excel 2002, a Microsoft ofereceu um esquema brilhante de proteção que, temporariamente, parecia frustrar os utilitários de quebra de senhas. A senha era fortemente encriptada. Vários meses depois do lançamento do Excel 2002, os programas de quebra de senhas tentaram combinações de força bruta. O software podia quebrar uma senha como *blue* em 10 minutos. Mas considerando uma senha de 24 caracteres, como *A6%kJJ542(9$GgU44#2drt8, o programa levaria 20 horas para descobrir a senha. Essa era uma dificuldade divertida a impor a outros programadores VBA, que potencialmente quebrariam seu código.

Mas a próxima versão do software de quebra de senhas era capaz de descobrir uma senha de 24 caracteres no Excel 2002 em 2 segundos. Quando testei meu projeto protegido por uma senha de 24 caracteres, o utilitário de senha rapidamente me informou que minha senha era XVII. Pensei que isso estava certamente errado, mas, depois de testar, descobri que a nova senha do projeto era XVII. Sim, essa última versão do software recorria a outra abordagem. Em vez de utilizar força bruta para quebrar a senha, esse software simplesmente gravava uma nova senha aleatória de quatro caracteres no projeto e salvava o arquivo.

Agora, isso causa um problema constrangedor para quem descobre a senha. Para se lembrar, o desenvolvedor colocou na parede a senha *A6%kJJ542(9$GgU44#2drt8. Mas, na versão quebrada do arquivo, a senha agora é XVII. Se houver um problema com o arquivo decifrado e ele for enviado de volta ao desenvolvedor, este não poderá mais abrir o arquivo. A única pessoa que conseguiria fazer isso é o programador na Estônia que escreveu o software de crack.

Não há desenvolvedores Excel VBA suficientes no mundo. Neste exato momento, há mais projetos do que programadores. Em meu círculo de amigos desenvolvedores, todos nós reconhecemos que perdemos perspectivas de negócio porque estamos muito ocupados com outros clientes.

Então, a situação de um desenvolvedor novato não é incomum. Ele escreve código para um cliente e, então, bloqueia o projeto VBA.

O cliente precisa fazer algumas alterações. O desenvolvedor original faz o trabalho. Algumas semanas mais tarde, mais algumas alterações são feitas e o desenvolvedor entrega o trabalho. Um mês mais tarde, o cliente precisa de mais correções. Ou o desenvolvedor está ocupado com outro projeto ou ganhando menos com esses trabalhos de manutenção e tem trabalhos mais lucrativos para fazer. O cliente tenta entrar em contato com o programador algumas vezes, então, percebendo que precisa do projeto corrigido, chama outro desenvolvedor — você!

Você obtém o código, que está protegido. Você descobre a senha e vê quem escreveu o código. Essa é uma decisão difícil. Você não tem interesse em roubar o cliente do rapaz. Você preferiria fazer esse trabalho e, então, fazer o cliente procurar o desenvolvedor original. Mas, por causa do hacking de senha, você criou agora uma situação em que os dois desenvolvedores têm senhas diferentes. Sua única escolha é remover completamente a senha.

Mais problemas com senhas

O esquema de senha para qualquer Excel da versão 2002 em diante é incompatível com o Excel 97. Se você protegeu código no Excel 2002, não poderá desbloquear o projeto no Excel 97. Muitas pessoas ainda utilizam o Excel 97. Como seu aplicativo é dado para mais funcionários de uma empresa, você invariavelmente encontra um funcionário com o Excel 97, e este, com certeza, aparecerá com algum erro de tempo de execução. Mas se você tiver bloqueado o projeto no Excel 2002 ou versão anterior, não poderá desbloqueá-lo no Excel 97 e, portanto, não poderá depurar no Excel 97.

Conclusão: bloquear o código causa tantos problemas que não vale a pena.

NOTA

Se você estiver utilizando uma combinação do Excel 2003 e 2007, as senhas são facilmente transferidas de um lado para outro. Isso ainda se aplica, mesmo se o arquivo for salvo como um arquivo XLSM e aberto no Excel 2003 utilizando o conversor de arquivo. Você pode modificar o código no Excel 2003, salvar o arquivo e *voltar com sucesso* para o Excel 2007.

Erros causados por versões diferentes

A Microsoft realmente aprimora o VBA a cada versão do Excel. A criação da tabela dinâmica foi significativamente aprimorada entre o Excel 97 e 2000 e novos recursos foram adicionados no Excel 2007. Certos recursos de gráficos foram aperfeiçoados entre o Excel 97 e o 2000 e os gráficos foram completamente reescritos no Excel 2007. O Excel começou suportando XML no Excel 2003 e parou de suportar a interatividade em páginas Web salvas no Excel 2007.

O parâmetro `TrailingMinusNumbers` era novo no Excel 2002. Se você escrever um código no Excel 2007 e o enviar a um cliente com o Excel 2000, esse usuário obterá um erro de compilação assim que tentar executar qualquer código no mesmo módulo do código com problemas. Considere esse aplicativo com dois módulos.

Module1 tem as macros ProcA, ProcB e ProcC. Module2 tem as macros ProcD e ProcE. Acontece que ProcE tem um método `ImportText` com o parâmetro `TrailingMinusNumbers`.

O cliente pode executar as macros ProcA e ProcB na máquina do Excel 2000 sem problemas, mas quando que tentar executar a macro ProcD, receberá um informe de erro de compilação em ProcD, porque o Excel tentará compilar todo o Module2 logo que o cliente executar o código nesse módulo. Isso pode ser incrivelmente enganoso: um erro relatado enquanto o cliente executa a macro ProcD é, na verdade, causado por um erro em ProcE.

Uma solução seria ter acesso a todas as versões suportadas do Excel, além do Excel 97, e testar o código em todas as versões. Note que o Excel 97 SR-2 era muito mais estável que as versões iniciais do Excel 97. Muitos clientes permanecem firmemente com o Excel 97, mas é frustrante quando você encontra alguém que não tem o Service Release estável.

Os usuários do Macintosh acreditarão que sua versão do Excel é idêntica ao Excel para Windows. A Microsoft prometeu compatibilidade de arquivos, mas essa promessa termina na interface com o usuário do Excel. O código VBA não é compatível entre o Windows e o Mac. Existe alguma semelhança, mas com uma diferença irritante. Certamente, tudo o que você fizer com a API do Windows não funcionará em um Mac.

Próximos passos

Este capítulo discutiu como tornar seu código mais à prova de falha para seus clientes. No Capítulo 26, "Personalizando a faixa para executar macros", você aprenderá a personalizar a faixa para permitir que seus clientes apreciem uma interface com o usuário profissional.

Personalizando a faixa para executar macros

26

NESTE CAPÍTULO

Fora com o velho, adote o novo

Uma das primeiras mudanças que você nota ao abrir o Excel 2007 é a nova barra de ferramentas da faixa. Os menus e as barras de ferramentas antigos não existem mais. E essa mudança não é apenas visual — o método de modificar controles de menu personalizados também mudou radicalmente. Uma das maiores vantagens desse novo método é que você não precisa mais se preocupar se sua barra de ferramentas personalizada permanecerá disponível depois que a pasta de trabalho for fechada, porque agora a barra de ferramentas personalizada faz parte do funcionamento interno da pasta de trabalho.

O objeto Barras de Comando original ainda funciona, mas os menus personalizados e as barras de ferramentas estão todos na guia, ou faixa, Suplementos. Se você tivesse comandos de menu personalizados, eles apareceriam no grupo Comandos de Menu, conforme mostrado na Figura 26.1. Na Figura 26.2, as barras de ferramentas personalizadas de duas pastas de trabalho diferentes aparecem juntas no grupo Barras de Ferramentas Personalizadas.

Se quiser modificar a faixa e adicionar sua própria guia, você precisa modificar o arquivo Excel, o que não é tão impossível quanto parece. Na realidade, o novo arquivo Excel é um arquivo compactado com utilitário zip, que contém vários arquivos e pastas. Tudo o que você precisa fazer é descompactar o arquivo zip, fazer as alterações e pronto. Ok, não é *tão* simples — são necessários mais alguns passos— mas não é impossível.

Antes de começarmos, vá ao menu do Office e selecione Opções do Excel, Avançado, Geral, e selecione Mostrar Erros da Interface de usuário em Suplementos. Isso permitirá que as mensagens de erro apareçam para que você possa resolvê-los em sua barra de ferramentas personalizada. Veja a seção "Solucionando mensagens de erro", mais adiante neste capítulo, para obter mais detalhes.

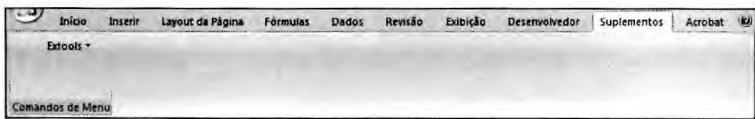



Figura 26.1
Os menus personalizados da versão anterior serão agrupados sob o grupo Comandos de Menu.

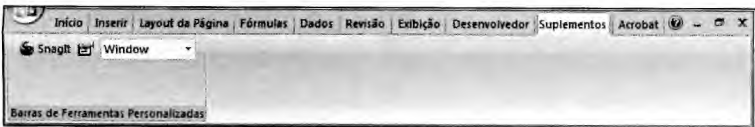



Figura 26.2
As barras de ferramentas personalizadas das versões anteriores do Excel aparecem no grupo Barras de Ferramentas Personalizadas.

> **ATENÇÃO**
>
> Diferentemente de programar no Editor do VB, você não terá assistência com a correção automática de caixa de letra (alta e baixa) e o código XML é muito específico. Observe a caixa de palavras específicas à XML, como `id`; utilizar `ID` irá gerar um erro.

Onde adicionar seu código: pasta e arquivo customui

Crie uma pasta chamada customui. Essa pasta conterá elementos de sua guia Faixa personalizada. Dentro da pasta, crie um arquivo de texto e nomeie-o customui.xml, como mostrado na Figura 26.3. Abra o arquivo XML em um editor de texto; o Bloco de Notas ou o WordPad funcionará

Figura 26.3
Crie um arquivo customui.xml dentro de uma pasta customui.

Insira a estrutura básica para o código XML, mostrado aqui, em seu arquivo XML. Para todo agrupamento de tag de abertura, como `<ribbon>`, há um tag de fechamento, `</ribbon>`:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="false">
    <tabs>

      <!--seus controles de faixa entram aqui -->

    </tabs>
  </ribbon>
</customUI>
```

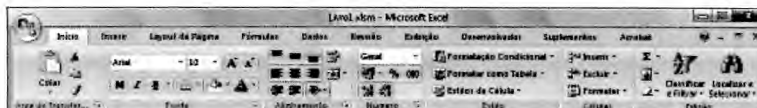
`startFromScratch` é opcional com um valor-padrão de `false`. É assim que você informa ao código que as outras guias no Excel não serão mostradas, apenas a sua. `True` significa mostrar somente a sua guia; `false` significa mostrar a sua guia e todas as outras.

> **ATENÇÃO**
>
> Observe a caixa das letras em `startFromScratch` — o *s* minúsculo no início, seguido pelo *F maiúsculo* em `From` e o *S maiúsculo* em `Scratch`. É crucial que você não se desvie disso.

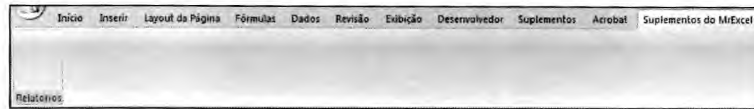
O `<!-- seus controles de faixa entram aqui -->` que você vê no código anterior é texto comentado. Apenas insira seus comentários entre `<!--` e `-->` e o programa ignorará a linha quando executar.

Criando a guia e o grupo

Antes de adicionar um controle a uma guia é necessário identificar a guia e o grupo. Uma guia armazena muitos controles diferentes, que podem ser agrupados, como o grupo Fonte na faixa Início, como mostrado na Figura 26.4.

Figura 26.4
Controles individuais são colocados em grupos em uma guia, a qual pode conter vários desses grupos.

Nomearemos nossa guia Suplementos do MrExcel e adicionaremos nela um grupo chamado Relatórios, como visto na Figura 26.5:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="false">
    <tabs>
      <tab id="CustomTab" label="Suplementos do MrExcel">
        <group id="CustomGroup" label="Relat&#243;rios">

          <!-- seus controles de faixa entram aqui -->

        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Figura 26.5
Adicione os tags Tab e Group ao código para criar uma guia e um grupo personalizados.

`Id` é um identificador único para o controle (nesse caso, a guia e grupo). `Label` é o texto que você quer exibir na faixa do controle especificado.

Adicionando um controle à faixa

Depois de configurar a faixa e o grupo, você pode adicionar os controles. Dependendo do tipo de controle, há diferentes atributos que você pode incluir no seu código XML. (Consulte a Tabela 26.1 para mais informações sobre os vários controles e seus atributos.)

O próximo código adiciona um botão de dimensionamento normal ao grupo Relatórios, definido para executar a sub chamada HelloWorld quando o botão for clicado (veja Figura 26.6):

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="false">
    <tabs>
       <tab id="CustomTab" label="Suplementos do MrExcel">
         <group id="CustomGroup" label="Relat&#243;rios">

           <button id="button1" label="Clique para executar"
               onAction="Module1.HelloWorld" size="normal"  />

         </group>
       </tab>
    </tabs>
  </ribbon>
</customUI>
```

`Id` é um identificador único para o botão de controle. `Label` é o texto que você quer que apareça em seu botão. `Size` é o tamanho do botão. `Normal` é o valor-padrão e a outra opção é `Large`. `onAction` é a sub, HelloWorld, a ser chamada quando o botão for clicado. A sub mostrada aqui entra em um módulo-padrão, Module1, na pasta de trabalho:

```
Sub HelloWorld(control As IRibbonControl)
MsgBox "Olá Mundo"
End Sub
```

Note o controle de argumento `As IRibbonControl`. Esse é o argumento-padrão para uma sub chamada por um controle de botão utilizando o atributo `onAction`. Consulte a Tabela 26.2 para os argumentos requeridos por outros atributos e controles.

Figura 26.6
Execute um programa com o clique de um botão em sua faixa personalizada.

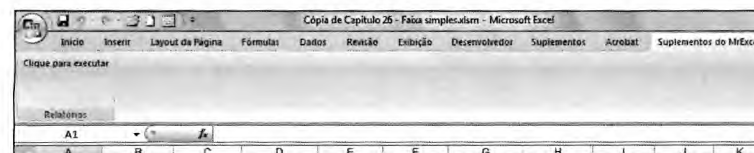



Tabela 26.1 Atributos de controle da faixa

Atributo	Tipo ou valor	Descrição
Descrição	Seqüência	Especifica o texto de descrição exibido em menus quando o atributo `itemSize` é configurado como `Large`.
`enabled`	`true`, `false`	Especifica se o controle está ativado.
getContent	Retorno de chamada	Recupera o conteúdo XML que descreve um menu dinâmico.
`getDescription`	Retorno de chamada	Obtém a descrição de um controle.
getEnabled	Retorno de chamada	Obtém o estado ativado de um controle.
`getImage`	Retorno de chamada	Obtém a imagem para um controle.
getImageMso	Retorno de chamada	Obtém o ícone de um controle predefinido utilizando o ID de controle.
`getItemCount`	Retorno de chamada	Obtém o número de itens a ser exibido em uma caixa de combinação, lista suspensa ou grupo.
getItemID	Retorno de chamada	Obtém o ID para um item específico em uma caixa de combinação, lista suspensa ou grupo.
`getItemImage`	Retorno de chamada	Obtém a imagem de uma caixa de combinação, lista suspensa ou grupo.
getItemLabel	Retorno de chamada	Obtém o rótulo de uma caixa de combinação, lista suspensa ou grupo.
`getItemScreentip`	Retorno de chamada	Obtém a Dica de Tela para uma caixa de combinação, lista suspensa ou grupo.
getItemSupertip	Retorno de chamada	Obtém a Dica de Tela Avançada para uma caixa de combinação, lista suspensa ou grupo.
`getKeytip`	Retorno de chamada	Obtém o KeyTip (dica da tecla aceleradora) para um controle.
getLabel	Retorno de chamada	Obtém o rótulo para um controle.
`getPressed`	Retorno de chamada	Obtém um valor que indica se um botão alternador foi ou não pressionado. Obtém um valor que indica se uma caixa de seleção está marcada ou desmarcada.
getScreentip	Retorno de chamada	Obtém a Dica de Tela para um controle.
`getSelectedItemID`	Retorno de chamada	Obtém o ID do item selecionado em uma lista suspensa ou grupo.
getSelectedItemIndex	Retorno de chamada	Obtém o índice do item selecionado em uma lista suspensa ou grupo.
`getShowImage`	Retorno de chamada	Obtém um valor que especifica a exibição ou não da imagem de controle.
getShowLabel	Retorno de chamada	Obtém um valor que define a exibição ou não do rótulo de controle.
`getSize`	Retorno de chamada	Obtém um valor que especifica o tamanho de um controle (normal ou grande).
getSupertip	Retorno de chamada	Obtém um valor que especifica a Dica de Tela Avançada para um controle.
`getText`	Retorno de chamada	Obtém o texto a ser exibido na parte de edição de uma caixa de texto ou edição.
getTitle	Retorno de chamada	Obtém o texto a ser exibido (em vez de uma linha horizontal) para um separador de menu.
`getVisible`	Retorno de chamada	Obtém um valor que especifica a visibilidade do controle.
Identificação	Seqüência	Um identificador único definido pelo usuário para o controle (mutuamente exclusivo com `idMso` e `idQ` — especifica apenas um desses valores).
`idMso`	Id de controle	O ID de controle predefinido (mutuamente exclusivo com `id` e `idQ` — especifica apenas um desses valores).
idQ	Id qualificado	ID qualificado de controle, prefixado com um identificador de espaço de nome (mutuamente exclusivo com `id` e `idMso` — especifica somente um desses valores).
`image`	Seqüência	Especifica uma imagem para o controle.
imageMso	Id de controle	Especifica um identificador para uma imagem predefinida.
`insertAfterMso`	Id de controle	Especifica o identificador para o controle predefinido depois que posicionar esse controle.

(continua)

Tabela 26.1 Atributos de controle da faixa

Atributo	Tipo ou valor	Descrição
insertAfterQ	Id qualificado	Especifica o identificador de um controle cuja propriedade `idQ` foi especificada depois desse controle ter sido posicionado.
`insertBeforeMso`	Id de controle	Especifica o identificador para o controle predefinido antes do qual posicionar esse controle.
insertBeforeQ	Id qualificado	Especifica o identificador de um controle cuja propriedade `idQ` foi especificada antes do qual posicionar esse controle.
`itemSize`	`large, normal`	Especifica o tamanho para os itens em um menu.
keytip	Seqüência	Especifica o KeyTip (dica da tecla aceleradora) para o controle.
`label`	Seqüência	Especifica o rótulo para o controle.
onAction	Retorno de chamada	Chamado quando o usuário clica no controle.
`onChange`	Retorno de chamada	Chamado quando o usuário insere ou seleciona texto em um caixa de edição ou caixa de combinação.
Dica de tela	Seqüência	Especifica a Dica de Tela do controle.
`showImage`	`true, false`	Especifica se a imagem do controle será exibida.
showItemImage	true, false	Especifica se a imagem será exibida em uma caixa de combinação, lista suspensa ou grupo.
`showItemLabel`	`true, false`	Especifica se o rótulo será mostrado em uma caixa de combinação, lista suspensa ou grupo.
showLabel	true, false	Especifica se o rótulo do controle será mostrado.
`size`	`large, normal`	Especifica o tamanho para o controle.
sizeString	Seqüência	Indica a largura para o controle especificando uma string, como `"xxxxxx.`
`supertip`	Seqüência	Especifica a Dica de Tela Avançada para o controle.
marca	Seqüência	Especifica texto definido pelo usuário.
`title`	Seqüência	Especifica o texto a ser exibido, em vez de uma linha horizontal, para um separador de menu.
`visible`	`true, false`	Especifica se o controle será visível.

Tabela 26.2 Argumentos de controle

Controle	Nome de retorno de chamada	Assinatura
Vários controles	`getDescription`	`Sub GetDescription(control as IRibbonControl, ByRef description)`
	`getEnabled`	`Sub GetEnabled(control As IRibbonControl, ByRef enabled)`
	`getImage`	`Sub GetImage(control As IRibbonControl, ByRef image)`
	`getImageMso`	`Sub GetImageMso(control As IRibbonControl, ByRef imageMso)`
	`getLabel`	`Sub GetLabel(control As IRibbonControl, ByRef label)`
	`getKeytip`	`Sub GetKeytip (control As IRibbonControl, ByRef label)`
	`getSize`	`sub GetSize(control As IRibbonControl, ByRef size)`
	`getScreentip`	`Sub GetScreentip(control As IRibbonControl, ByRef screentip)`
	`getSupertip`	`Sub GetSupertip(control As IRibbonControl, ByRef screentip)`
	`getVisible`	`Sub GetVisible(control As IRibbonControl, ByRef visible)`

(continua)

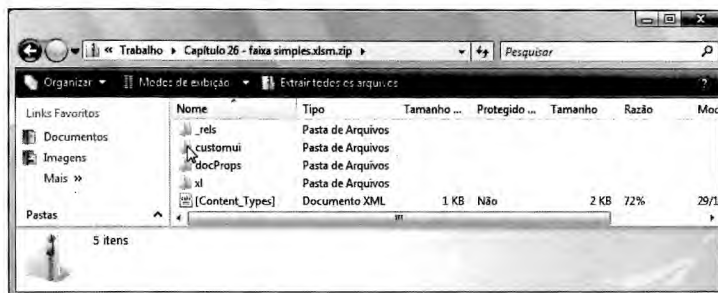
Tabela 26.2 Argumentos de controle (*cont.*)

Controle	Nome de retorno de chamada	Assinatura
button	getShowImage	Sub GetShowImage (control As IRibbonControl, ByRef showImage)
	getShowLabel	Sub GetShowLabel (control As IRibbonControl, ByRef showLabel)
	onAction	Sub OnAction(control As IRibbonControl)
checkBox	getPressed	Sub GetPressed(control As IRibbonControl, ByRef returnValue)
	onAction	Sub OnAction(control As IRibbonControl, pressed As Boolean)
comboBox	getItemCount	Sub GetItemCount(control As IRibbonControl, ByRef count)
	getItemID	Sub GetItemID(control As IRibbonControl, index As Integer, ByRef id)
	getItemImage	Sub GetItemImage(control As IRibbonControl, index As Integer, ByRef image)
	getItemLabel	Sub GetItemLabel(control As IRibbonControl, index As Integer, ByRef label)
	getItemScreenTip	Sub GetItemScreenTip(control As IRibbonControl, index As Integer, ByRef screentip)
	getItemSuperTip	Sub GetItemSuperTip (control As IRibbonControl, index As Integer, ByRef supertip)
	getText	Sub GetText(control As IRibbonControl, ByRef text)
	onChange	Sub OnChange(control As IRibbonControl, text As String)
customUI	loadImage	Sub LoadImage(imageId As string, ByRef image)
	onLoad	Sub OnLoad(ribbon As IRibbonUI)
dropDown	getItemCount	Sub GetItemCount(control As IRibbonControl, ByRef count)
	getItemID	Sub GetItemID(control As IRibbonControl, index As Integer, ByRef id)
	getItemImage	Sub GetItemImage(control As IRibbonControl, index As Integer, ByRef image)
dropDown	getItemLabel	Sub GetItemLabel(control As IRibbonControl, index As Integer, ByRef label)
	getItemScreenTip	Sub GetItemScreenTip(control As IRibbonControl, index As Integer, ByRef screenTip)
	getItemSuperTip	Sub GetItemSuperTip (control As IRibbonControl, index As Integer, ByRef superTip)
	getSelectedItemID	Sub GetSelectedItemID(control As IRibbonControl, ByRef index)
	getSelectedItemIndex	Sub GetSelectedItemIndex(control As IRibbonControl, ByRef index)
	onAction	Sub OnAction(control As IRibbonControl, selectedId As String, selectedIndex As Integer)
dynamicMen	getContent	Sub GetContent(control As IRibbonControl, ByRef content)
editBox	getText	Sub GetText(control As IRibbonControl, ByRef text)
	onChange	Sub OnChange(control As IRibbonControl, text As String)
gallery	getItemCount	Sub GetItemCount(control As IRibbonControl, ByRef count)
	getItemHeight	Sub getItemHeight(control As IRibbonControl, ByRef height)
	getItemID	Sub GetItemID(control As IRibbonControl, index As Integer, ByRef id)
	getItemImage	Sub GetItemImage(control As IRibbonControl, index As Integer, ByRef image)
	getItemLabel	Sub GetItemLabel(control As IRibbonControl, index As Integer, ByRef label)
	getItemScreenTip	Sub GetItemScreenTip(control As IRibbonControl, index as Integer, ByRef screen)
gallery	getItemSuperTip	Sub GetItemSuperTip (control As IRibbonControl, index as Integer, ByRef screen)
	getItemWidth	Sub getItemWidth(control As IRibbonControl, ByRef width)
	getSelectedItemID	Sub GetSelectedItemID(control As IRibbonControl, ByRef index)
	getSelectedItemIndex	Sub GetSelectedItemIndex(control As IRibbonControl, ByRef index)
	onAction	Sub OnAction(control As IRibbonControl, selectedId As String, selectedIndex As Integer)
menuSeparator	getTitle	Sub GetTitle (control As IRibbonControl, ByRef title)
toggleButton	getPressed	Sub GetPressed(control As IRibbonControl, ByRef returnValue)
	onAction	Sub OnAction(control As IRibbonControl, pressed As Boolean)

Acessando a estrutura de arquivo

Os novos tipos de arquivo Excel são, na realidade, arquivos compactados com um utilitário zip, que contém vários arquivos e pastas para criar a pasta de trabalho e planilhas que você vê quando abre a pasta de trabalho. Para visualizar essa estrutura, renomeie o arquivo, adicionando uma extensão .zip ao fim do nome de arquivo. Por exemplo, se o nome de arquivo for Capítulo 26 – faixa simples. xlsm, renomeie-o para Capítulo 26 – faixa simples.xlsm.zip. Em seguida, você pode utilizar o utilitário zip para acessar as pastas e arquivos contidos no arquivo.

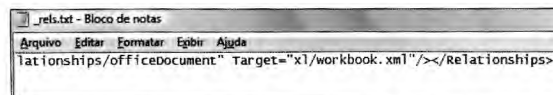
Copie no arquivo zip sua pasta e arquivo customui, como mostrado na Figura 26.7. Depois de colocá-los no arquivo XLSM, precisamos deixar que o resto do arquivo Excel saiba que eles estão lá e qual o seu propósito. Para fazer isso, modificamos o arquivo RELS.

Figura 26.7
Utilizando um utilitário zip, abra o arquivo XLSM e o copie na pasta e arquivo customui.

Entendendo o arquivo RELS

O arquivo RELS, localizado na pasta _rels, contém os vários relacionamentos do arquivo Excel. Extraia esse arquivo do zip e abra-o com um editor de textos.

O arquivo já contém relacionamentos existentes que não queremos alterar. Em vez disso, precisamos adicionar um para a pasta customui. Role pela direita da linha `<Relationships` e coloque o cursor antes do tag `</Relationships>`, como mostrado na Figura 26.8. Insira a seguinte sintaxe:

```
<Relationship Id="rAB67989"
➥Type="http://schemas.microsoft.com/office/2006/relationships/ui/extensibility"
➥Target="customui/customui.xml"
```

Figura 26.8
Coloque o cursor no ponto correto para inserir o relacionamento de faixa personalizada.

ATENÇÃO

Mesmo que o código anterior apareça como três linhas neste livro, ele deve aparecer como uma única linha no arquivo RELS. Se quiser inseri-lo como três linhas separadas, não separe as linhas dentro das strings citadas. Os exemplos anteriores são quebras corretas. Uma quebra incorreta da terceira linha, por exemplo, seria assim:

```
Target = "customui/
customui.xml"
```

Observe que o Excel mesclará as três linhas separadas acima em uma única linha, quando a pasta de trabalho for aberta.

`Id` é qualquer string única para identificar o relacionamento. Se o Excel tiver um problema com a string inserida, você pode mudar isso ao abrir o arquivo. (Veja a seção de solução de problemas "O Excel localizou conteúdo ilegível", mais adiante neste capítulo, para informações adicionais.) `Target` é a pasta e o arquivo customui.

Salve as alterações e compacte o arquivo RELS utilizando o utilitário zip.

Renomeando o arquivo Excel e abrindo a pasta de trabalho

Renomeie o arquivo Excel com o nome original removendo a extensão .zip. Abra sua pasta de trabalho. Se aparecer alguma mensagem de erro, consulte a seção "Solucionando problemas de mensagens de erro" neste capítulo.

RibbonCustomizer

Pode ser um pouco demorado realizar todos os passos para adicionar uma faixa personalizada, especialmente se você cometer pequenos erros e continuar renomeando a pasta de trabalho, abrindo o arquivo zip, extraindo o arquivo, modificando-o, recompactando-o com o utilitário zip, renomeando-o e testando-o. Para ajudar nisso, Patrick Schmid da pschmid.net criou o RibbonCustomizer, que faz a maioria dessas ações para você dentro de sua interface. Vá para `http://pschmid.net/ office2007/ribboncustomizer/index.php` para obter mais informações sobre essa ferramenta.

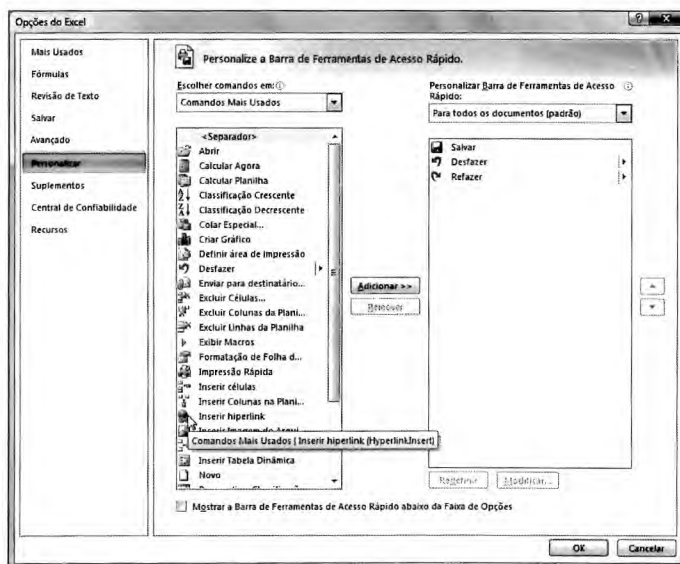
Utilizando imagens em botões

A imagem que aparece em um botão pode ser uma imagem da biblioteca de ícones do Microsoft Office ou uma imagem personalizada que você cria e inclui dentro da pasta customui da pasta de trabalho. Com uma boa imagem de ícone, você pode ocultar o rótulo do botão, mas ainda ter uma faixa amigável com imagens que são autoexplicativas.

Ícones do Microsoft Office

Lembra-se, nas versões anteriores do Excel, de que quando queria reutilizar um ícone de um botão Excel, você tinha de identificar o `faceid`? Era um pesadelo fazer isso à mão, embora, felizmente, houvesse muitas ferramentas para ajudar a recuperar informações. Bem, a Microsoft deve ter ouvido os gritos de agonia, porque ela tornou mais fácil a reutilização dos ícones. Não apenas isso: em vez de algum número sem sentido, a Microsoft forneceu um texto mais fácil de entender!

Escolha menu Office, Opções do Excel, Personalizar. Coloque o cursor sobre qualquer comando de menu na lista e uma Dica de Tela será exibida, fornecendo mais informações sobre o comando. Incluído na mesma extremidade, entre parênteses, há o nome da imagem, como mostrado na Figura 26.9.

Figura 26.9
Colocar o cursor sobre um comando, como Inserir Hiperlink, apresenta o nome de ícone, HyperlinkInsert.

Para colocar uma imagem em nosso botão, precisamos voltar para o arquivo customui e 'dizer' ao Excel o que queremos. O código a seguir utiliza o ícone HyperlinkInsert para o botão HelloWorld e também oculta o rótulo, como mostrado na Figura 26.10. Observe que o nome do ícone diferencia letras maiúsculas de minúsculas:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="false">
    <tabs>
      <tab id="CustomTab" label="Suplementos do MrExcel">
        <group id="CustomGroup" label="Relat&#243;rios">

          <button id="button1" label="Clique para executar"
            onAction="Module1.HelloWorld" imageMso="HyperlinkInsert"
            showLabel = "false" />

        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Figura 26.10
Você pode aplicar a imagem de qualquer ícone do Microsoft Office ao seu botão personalizado.

> **DICA**
> Você não está limitado apenas aos ícones disponíveis no Excel. Pode utilizar o ícone para qualquer aplicativo Microsoft Office instalado. Você pode descarregar uma pasta de trabalho da Microsoft com vários grupos mostrando os ícones disponíveis (e seus nomes) do seguinte endereço: www.microsoft.com/downloads/ details.aspx?familyid=12b99325-93e8-4ed4-8385-74d0f7661318.

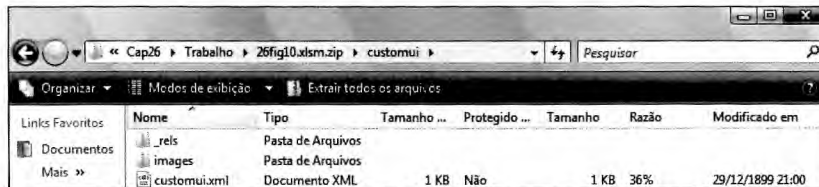
Imagens de ícone personalizadas

E se a biblioteca de ícones não tiver o ícone que você está procurando? Você pode criar seu próprio arquivo de imagem e modificar a faixa para utilizá-lo:

1. Crie uma pasta chamada images na pasta customui. Coloque a imagem nessa pasta.

2. Crie uma pasta chamada _rels na pasta customui. Crie um arquivo de texto chamado customui.xml.rels nessa nova pasta, como mostrado na Figura 26.11. Coloque o seguinte código no arquivo. Note que o `Id` para o relacionamento da imagem é o nome do arquivo de imagem, mrexcellogo:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/
➥relationships"><Relationship Id="mrexcellogo"
➥Type=http://schemas.openxmlformats.org/officeDocument/2006/relationships/image
➥Target="images/mrexcellogo.jpg"/></Relationships>
```

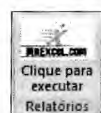
Figura 26.11
Crie um _rels e uma pasta de imagens dentro da pasta customui para armazenar arquivos relevantes à imagem personalizada.

3. Abra o arquivo customui.xml e adicione o atributo de imagem ao controle, como mostrado aqui. Salve e feche o arquivo:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="false">
    <tabs>
      <tab id="CustomTab" label="Suplementos do MrExcel">
        <group id="CustomGroup" label="Relat&#243;rios">

          <button id="button1" label="Clique para executar"
            onAction="Module1.HelloWorld" image="mrexcellogo"
            size="large" />

        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

4. Abra o arquivo [Content_Types].xml e adicione o seguinte bem no fim do arquivo, mas antes de </Types>:

```
<Default Extension="jpg" ContentType="application/octet-stream"/>
```

5. Salve as alterações, renomeie a pasta e abra a pasta de trabalho. A imagem personalizada aparecerá no botão, como mostrado na Figura 26.12.

Figura 26.12
Com mais algumas alterações no customui, você pode adicionar uma imagem personalizada a um botão.

Estudo de caso

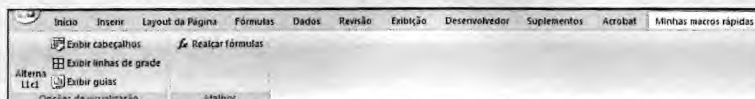
Convertendo uma barra de ferramentas personalizada do Excel 2003 para o Excel 2007

Você tem uma pasta de trabalho e uma barra de ferramentas personalizadas projetadas no Excel 2003 com vários botões. Agora você está pronto para transferir para o Excel 2007. Quando abrir a pasta de trabalho no 2007, a barra de ferramentas não aparecerá na faixa Suplementos porque não foi projetada com o VBA; é uma barra de ferramentas personalizada criada manualmente.

Depois de salvar a pasta de trabalho como um arquivo XLSM, crie o arquivo customui, como mostrado aqui. A guia é chamada My Quick Macros e tem dois grupos: Opções de Visualização e Atalhos (note que, no código, os caracteres acentuados ou especiais são convertidos em entidades XML):

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="false">
    <tabs>
      <tab id="customMacros" label="Minhas macros r&#225;pidas">
        <group id="customview" label="Op&#231;&#245;es de Visualiza&#231;&#227;o">

          <button id="btn_r1c1" label="Alterna L1c1"
            onAction="mod_2007.myButtons" />

          <button id="btn_Headings" label="Exibir cabe&#231;alhos"
            onAction="mod_2007.myButtons" imageMso = "TableStyleClear"/>

          <button id="btn_gridlines" label="Exibir linhas de grade"
            onAction="mod_2007.myButtons" imageMso = "BordersAll"/>

          <button id="btn_tabs" label="Exibir guias"
            onAction="mod_2007.myButtons" imageMso = "Connections"/>
        </group>

        <group id="customshortcuts" label="Atalhos">

          <button id="btn_formulas" label="Real&#231;ar f&#243;rmulas"
            onAction="mod_2007.myButtons" imageMso = "FunctionWizard"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
```

Depois de atualizar o arquivo RELS (veja a seção "Entendendo o arquivo RELS" anteriormente neste capítulo para obter mais detalhes), abra a pasta de trabalho para ver a nova guia, como mostrado na Figura 26.13.

Figura 26.13
Recrie a barra de ferramentas do Excel 2003 no Excel 2007 como sua própria faixa.

Está na hora de atualizar o código na pasta de trabalho. Você notará que `onAction` na pasta customui aponta inteiramente para a mesma sub, mod_2007.myButtons, em vez de cada uma ter uma chamada personalizada. Como todos os controles são do mesmo tipo, botões, e têm o mesmo tipo de argumento, `iRibbonControl`, tiramos proveito desses fatos. Crie uma única sub, `myButtons`, em um módulo chamado mod_2007 para tratar todas as chamadas de botão utilizando `Select Case` para gerenciar os IDs de cada botão:

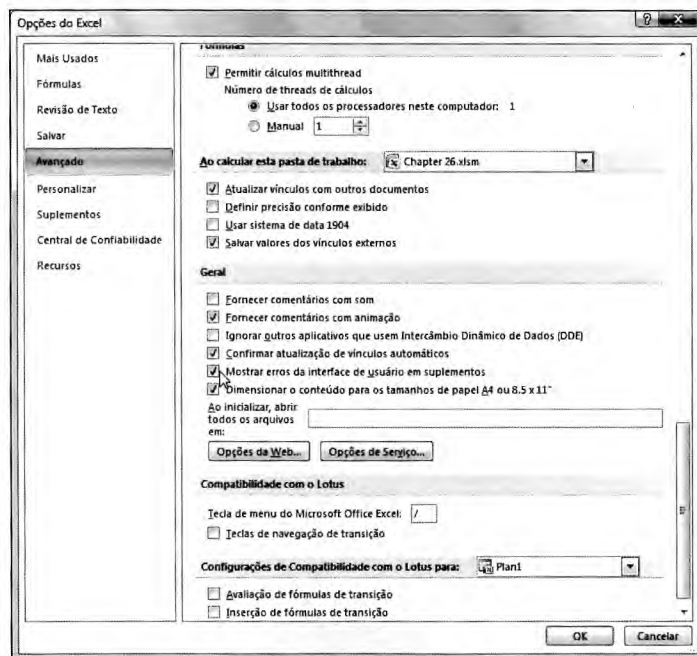
```
Sub myButtons(control As IRibbonControl)
Select Case control.ID
    Case Is = "btn_r1c1"
        SwitchR1C1
    Case Is = "btn_Headings"
        ShowHeaders
    Case Is = "btn_gridlines"
        ShowGridlines
    Case Is = "btn_tabs"
        ShowTabs
    Case Is = "btn_formulas"
        GoToFormulas
    End Select
End Sub
```

Os `control.IDs` são os ids atribuídos a cada botão no arquivo customui.xml. A ação dentro de cada instrução `Case` é uma chamada para a sub desejada. Eis um exemplo de uma das subs sendo chamada, `ShowHeaders`. É a mesma sub que estava na pasta de trabalho original 2003:

```
Sub ShowHeaders()
If ActiveWindow.DisplayHeadings = False Then
    ActiveWindow.DisplayHeadings = True
Else
    ActiveWindow.DisplayHeadings = False
End If
End Sub
```

Solucionando problemas de mensagens de erro

Para ver as mensagens de erro geradas por uma faixa personalizada, vá para o menu Office e selecione Opções do Excel, Avançado, Geral, e verifique Mostrar Erros da Interface do Usuário em Suplementos, como mostrado na Figura 26.14.

Figura 26.14
Marque a opção Mostrar Erros da Interface do Usuário em Suplementos para permitir que as mensagens de erro personalizadas da faixa apareçam e ajudem a solucionar problemas.

O atributo 'Attribute Name' no elemento 'customui Ribbon' não está definido em DTD/Schema

Como você notou na seção "Onde adicionar seu código: pasta e arquivo customui" deste capítulo, o uso de caixa alta e baixa dos atributos é muito especial. Se um atributo estiver com a "caixa errada", o erro mostrado na Figura 26.15 pode ocorrer. O código customui.xml que gerou o erro tinha a seguinte linha:

```
<ribbon startfromscratch="false">
```

Em vez de `startFromScratch`, o código continha `startfromscratch` (todas as letras minúsculas). A mensagem de erro ajuda até mesmo a restringir o problema nomeando o atributo que está com um problema.

Figura 26.15
Atributos com incorreções na indicação de maiúsculas e minúsculas podem gerar erros. Leia a mensagem de erro cuidadosamente; isso pode ajudar a rastrear o problema.

Caractere de nome qualificado ilegal

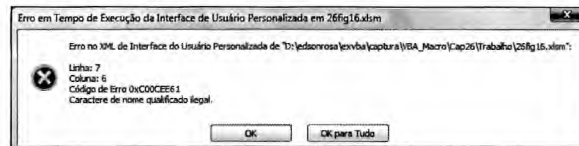
Para cada < de abertura, você precisa de um > de fechamento. Se você se esquecer de um > de fechamento, o erro mostrado na Figura 26.16 pode aparecer. A mensagem de erro não é de modo algum específica, mas fornece um número de linha e coluna onde um pro-

blema ocorreu. Entretanto, não é o ponto real em que o > ausente estaria. Em vez disso, é o começo da próxima linha. Você precisará revisar o código para localizar o erro, mas terá uma idéia de onde iniciar. O código a seguir no customui.xml gerou o erro:

```
<tab id="CustomTab" label="Suplementos do MrExcel">
  <group id="CustomGroup" label="Relat&#243;rios"
    <button id="button1" label="Clique para executar"
      onAction="Module1.HelloWorld" image="mrexcellogo"
      size="large" />
```

Note o > ausente para a linha de grupo (segunda linha de código). A linha deveria ter sido esta:

```
<group id="CustomGroup" label="Relat&#243;rios">
```

Figura 26.16
Para cada < de abertura você precisa de um > de fechamento.

O elemento 'Nome do Tag customui' é inesperado de acordo com o modelo de conteúdo do elemento pai 'Nome do Tag customui'

Se sua estrutura estiver na ordem errada, como o tag de grupo colocado antes do tag de guia, como mostrado aqui, aparecerá uma cadeia de erros, a começar por esta mostrada na Figura 26.17:

```
<group id="CustomGroup" label="Relat&#243;rios">
  <tab id="CustomTab" label="Suplementos do MrExcel">
```

Figura 26.17
Um erro em uma linha pode resultar em uma string de mensagens de erro porque as outras linhas são agora consideradas fora de ordem.

O Excel localizou conteúdo ilegível

A Figura 26.18 mostra uma mensagem genérica para diferentes tipos de problemas que o Excel pode encontrar. Se você clicar em Sim, então receberá a mensagem mostrada na Figura 26.19. Se clicar em Não, a pasta de trabalho não será aberta. Mas ao criar a faixa, descobri que ela aparecia com mais freqüência quando o Excel não gostava do id de relacionamento que eu havia atribuído ao relacionamento customui no arquivo .RELS. O interessante é que, se você clicar em Sim, o Excel atribuirá um novo arquivo id e, da próxima vez que você abrir o arquivo, o erro não deverá aparecer.

Relacionamento original:

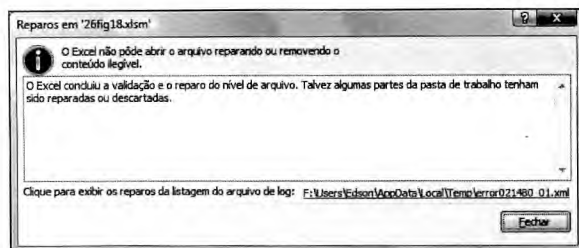
```
<Relationship Id="rId3"
➥Type=http://schemas.microsoft.com/office/2006/relationships/ui/extensibility
➥Target="customui/customui.xml"/>
```

O Excel modificou o relacionamento:

```
<Relationship Id="rE1FA1CF0-6CA9-499E-9217-90BF2D86492F"
➥Type="http://schemas.microsoft.com/office/2006/relationships/ui/extensibility"
➥Target="customui/customui.xml"/>
```

O erro também aparecerá se, no arquivo RELS, você dividir a linha de relacionamento dentro de uma string citada, como desaconselhado na seção "Entendendo o arquivo RELS", anteriormente neste capítulo. Nesse caso, o Excel não corrigirá o arquivo e você mesmo deve fazer a correção.

Figura 26.18
Essa mensagem de erro bastante genérica pode aparecer por muitas razões. Clique em Sim para tentar reparar o arquivo.

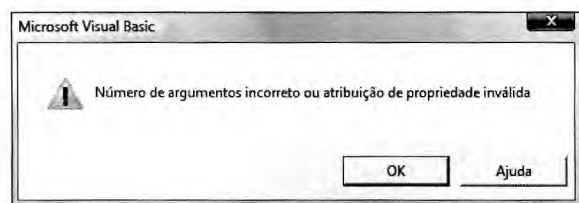
Figura 26.19
O Excel permitirá que você saiba se ele foi bem-sucedido na correção do arquivo.

Número errado de argumentos ou atribuição de propriedade inválida

Se houver um problema com a sub que estiver sendo chamada pelo controle, talvez você veja o erro ilustrado na Figura 26.20 ao acessar sua faixa. Por exemplo, a sub `onAction` de um botão requer um único argumento `IRibbonControl`, como o seguinte:

```
Sub HelloWorld(control As IRibbonControl)
```

Seria incorreto ocultar o argumento, como mostrado aqui:

```
Sub HelloWorld()
```

Figura 26.20
É importante que as sub que estiverem sendo chamadas por seus controles tenham os argumentos adequados. Consulte a Tabela 26.2 para os vários argumentos de controle.

Nada acontece

Se você abrir sua pasta de trabalho modificada, a faixa não aparecer, mas não houver nenhuma mensagem de erro, marque duas vezes o arquivo RELS. É possível que você tenha esquecido de atualizá-lo com o relacionamento requerido para seu custumui.xml.

Outras maneiras de executar uma macro

As faixas personalizadas são as melhores maneiras de executar uma macro; mas se você tiver apenas duas macros para executar, modificar o arquivo pode ser um pouco trabalhoso. No Excel 2003, era fácil fazer um cliente invocar uma macro ao acessar Ferramentas, Macro, Macros, selecionar a macro da caixa de diálogo Macro e clicar no botão Executar (embora fosse um pouco não-profissional). Mas agora essa opção está muito longe de ser conveniente.

Atalho de teclado

A maneira mais fácil de executar uma macro é atribuir um atalho de teclado a uma macro. Na caixa de diálogo Macro (faixa Desenvolvedor, clique em Macros ou pressione Alt+F8), selecione a macro e clique em Opções. Atribua uma tecla de atalho à macro. A Figura 26.21 mostra o atalho Ctrl+Shift+C sendo atribuído à macro Clean1stCol. Você, então, pode postar visivelmente uma nota na planilha para lembrar o cliente de pressionar Ctrl+Shift+C para limpar a primeira coluna.

Figura 26.21
A maneira mais simples de permitir que um cliente execute uma macro é atribuir uma tecla de atalho à macro. Ctrl+Shift+C agora executa a macro Clean1stCol.

ATENÇÃO

Seja cuidadoso ao atribuir os atalhos de teclado. Muitas das teclas já estão mapeadas para atalhos importantes do Windows. Se você atribuísse uma macro a Ctrl+C, qualquer pessoa que utilizasse esse atalho para copiar a seleção para a área de transferência ficaria frustrado quando o aplicativo fizesse qualquer coisa em resposta. Em geral, as letras E, J, M, Q e T são boas escolhas porque, como no Excel 2007, elas ainda não foram atribuídas ao menu do Excel de combinações de atalho "Ctrl+". O atalho Ctrl+L costumava estar disponível, mas essa combinação é agora utilizada para criar uma tabela no Excel 2007.

Anexe uma macro a um botão de comando

Dois tipos de botões podem ser incorporados à planilha: a forma tradicional de botão, encontrada no botão de comando Controles de Formulário, ou o botão de comando Controle ActiveX (ambos podem ser acessados na faixa Desenvolvedor, na opção Inserir).

Para adicionar um botão Controle de Formulário com uma macro à planilha, siga estes passos:

1. Na guia Desenvolvedor, clique no botão Inserir e selecione o controle de botão na seção Controles de Formulário, como mostrado na Figura 26.22.

Figura 26.22
Os Controles de Formulário estão localizados no ícone Inserir na faixa Desenvolvedor.

2. Coloque o cursor na planilha em que quer inserir o botão e, então, clique e arraste para criar a forma de novo botão.
3. Quando você soltar o botão do mouse, a caixa de diálogo Atribuir Macro aparecerá. Selecione uma macro a ser atribuída para o botão e escolha OK.
4. Destaque o texto do botão e digite um novo texto descritivo.
5. Para mudar a cor, a fonte e outros aspectos do botão, clique com o botão direito do mouse no botão e escolha Controle de Formato no menu suspenso.
6. Para reatribuir uma nova macro ao botão, clique com o botão direito do mouse no botão e escolha Atribuir Macro no menu suspenso.

ATENÇÃO

Há duas versões da caixa de diálogo Controle de Formato. Uma versão oferece sete guias com configurações para Fonte, Alinhamento, Tamanho, Proteção, Propriedades, Margens e Web. A segunda versão oferece apenas uma guia com configurações para Fonte.

A versão que você deve usar é baseada em algo muito sutil. Quando clicar com o botão direito do mouse no controle, a aparência da borda de seleção pode ser de linhas diagonais, como mostrado na Figura 26.23, ou de pontos, como mostrado na Figura 26.24. A borda de seleção de linha diagonal tende a aparecer logo depois da edição do texto do botão e leva à caixa de diálogo Controle de Formato de Fonte somente. A borda de seleção de pontos resulta na caixa de diálogo Controle de Formato completa, com todos os recursos.

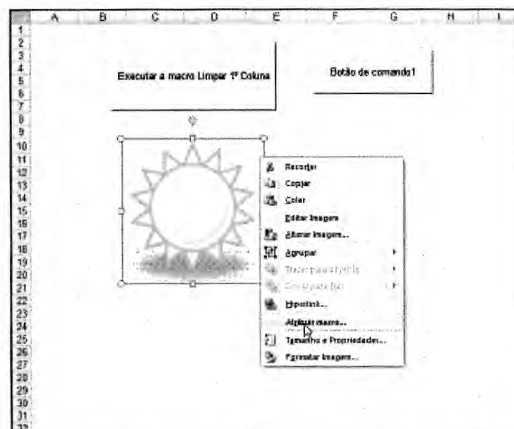
Figura 26.23
A borda de seleção de linha diagonal resulta em uma caixa de diálogo Controle de Formato inferior.

continua

Figura 26.24
A borda de seleção de pontos leva à versão completa da caixa de diálogo Controle de Formato.

Se você clicar com o botão direito do mouse e as linhas diagonais aparecerem como a borda de seleção, você deve primeiro clicar com o botão esquerdo do mouse na borda de seleção de linha diagonal para mudá-la para pontos. Você, então, pode clicar com o botão direito do mouse novamente e escolher Controle de Formato.

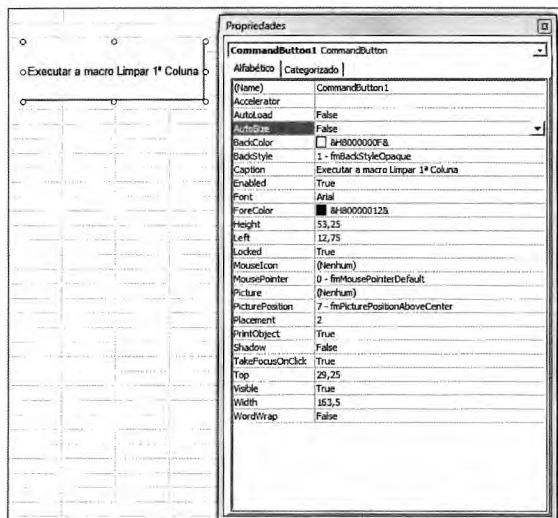
Esse método atribuiu uma macro a um objeto que parece um botão. Você também pode atribuir uma macro a qualquer objeto de desenho na planilha. Para atribuir uma macro a uma AutoForma, clique com o botão direito do mouse na forma e selecione Atribuir Macro, como mostrado na Figura 26.25.

Figura 26.25
As macros podem ser atribuídas a qualquer objeto de desenho na planilha.

Prefiro esse método porque posso adicionar facilmente um objeto de desenho com código de macro e utilizar a propriedade `OnAction` para atribuir uma macro ao objeto. Há uma grande desvantagem no uso desse método: se você atribuir uma macro que existe em outra pasta de trabalho e esta pasta for salva e fechada, o Excel irá alterar a sub `OnAction` do objeto a ser codificada manualmente para uma pasta específica.

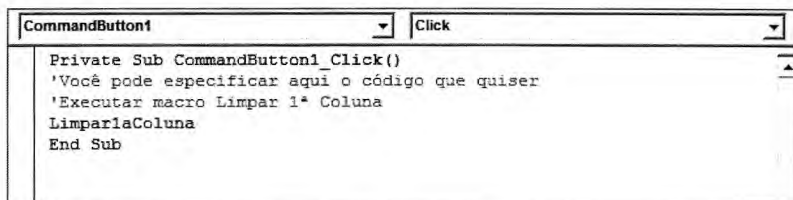
Anexe uma macro a um controle ActiveX

Os controles ActiveX são mais novos que os Controles de Formulário e sua configuração é ligeiramente mais complicada. Em vez de simplesmente atribuir uma macro ao botão, você terá um procedimento `button_click` em que pode chamar outra macro ou ter o código de macro realmente incorporado no procedimento `button_click`. Siga estes passos:

1. Na guia Desenvolvedor, clique no botão Inserir e selecione o ícone Botão de Comando da seção Controles ActiveX da caixa de ferramentas suspensa Controle.
2. Desenhe a forma de um botão na planilha como descrito no passo 2 para o botão de Formulários.
3. Para formatar o botão, clique com o botão direito do mouse no botão e selecione Propriedades ou selecione Propriedades da faixa Desenvolvedor. Agora você pode ajustar a legenda e a cor do botão na janela Propriedades, como mostrado na Figura 26.26. Se nada acontecer quando clicar com o botão direito do mouse no botão, entre no modo Design clicando no botão Modo de Design na faixa Desenvolvedor.

Figura 26.26
Clicar no ícone Propriedades abre a janela Propriedades, na qual você pode ajustar muitos aspectos do botão ActiveX.

> **NOTA**
> Há um aspecto irritante dessa janela Propriedades: ela é enorme e ocupa uma grande parte de sua planilha. Por fim, se quiser utilizar a planilha, será necessário fechar essa janela. Quando fazemos isso, também ocultamos a janela Propriedades do Editor do VB. Você pode ver que ela está ausente na Figura 26.27. Eu preferiria poder fechá-la sem afetar meu ambiente Editor do VB.

4. Para atribuir uma macro ao botão, clique no botão Exibir Código no grupo Controles da faixa Desenvolvedor. Isso cria um novo procedimento no painel de código para a planilha atual. Digite o código ou o nome da macro que você quer executar nesse procedimento. A Figura 26.27 mostra o código do botão. Esse código aparece no painel de código para a planilha.

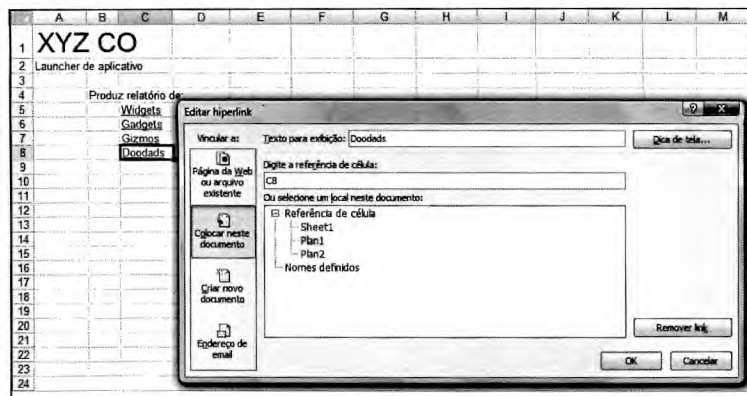
Figura 26.27
Clique no botão Exibir Código na barra de ferramentas Caixa de Ferramentas de Controle para abrir a macro para esse botão.

Executando uma macro a partir de um hiperlink

Utilizando um truque é possível executar uma macro a partir de um hiperlink. Como muitos clientes estão acostumados a clicar em um hiperlink a fim de realizar uma ação, esse método talvez seja mais intuitivo para seus clientes.

O truque é configurar hiperlinks provisórios que simplesmente levam para o mesmo lugar. Selecione uma célula e da faixa Inserir, selecione Hiperlink e pressione Ctrl+K. Na caixa de diálogo Inserir Hiperlink, clique em Colocar neste Documento. A Figura 26.28 mostra uma planilha com quatro hiperlinks, cada um apontando para a própria célula.

Quando um cliente clica em um hiperlink, você pode interceptar essa ação e executar qualquer macro utilizando o evento `FollowHyperlink`. Insira o seguinte código no módulo de código da planilha:

```
Private Sub Worksheet_FollowHyperlink(ByVal Target As Hyperlink)
Select Case Target.TextToDisplay
    Case "Widgets"
        RunWidgetReport
    Case "Gadgets"
        RunGadgetReport
    Case "Gizmos"
        RunGizmoReport
    Case "Doodads"
        RunDooDadReport
End Select
End Sub
```

Figura 26.28
Para executar uma macro de um hiperlink, você tem de criar hiperlinks provisórios que levam para suas próprias *células*. Então, *utilizando* uma macro de rotina de tratamento de evento no painel de código da planilha, você pode interceptar o hiperlink e executar qualquer macro.

Próximos passos

A partir das faixas personalizadas para botões simples ou hiperlinks, há muitas maneiras de assegurar que seus clientes nunca precisem ver a caixa de diálogo Macro. No Capítulo 27, "Criando suplementos", você aprenderá a 'empacotar' suas macros em suplementos que podem ser facilmente distribuídos para outros.

Criando suplementos

27

Utilizando o VBA, você pode criar arquivos de suplementos-padrão para seus clientes utilizar. Depois que o cliente instala o suplemento no PC, o programa estará disponível para o Excel e carregará automaticamente toda vez que ele abrir o Excel.

Este capítulo discute os suplementos-padrão. Esteja ciente de que há dois outros tipos de suplementos: Os suplementos COM e os suplementos DLL. Nenhum desses pode ser criado com o VBA. Para criar esses tipos de suplementos, você precisa do Visual Basic .NET ou do Visual C++.

Características dos suplementos-padrão

Se for distribuir seus aplicativos, talvez você queira empacotá-los como suplementos. Em geral, quando salvo com uma extensão .xlam para o Excel 2007 ou uma extensão .xla para o Excel 97-2003, o suplemento oferece várias vantagens:

- Normalmente, os clientes podem pular seu código `Workbook_Open` mantendo a tecla Shift pressionada enquanto abrem a pasta de trabalho. Com um suplemento, eles não podem pular o código `Workbook_Open` dessa maneira.
- Depois que a caixa de diálogo Suplementos for utilizada para instalar um suplemento (selecione o Ícone do Office, Opções do Excel, Suplementos, Gerenciar: Suplementos do Excel, Ir), o suplemento sempre estará carregado e disponível.
- Mesmo se o nível de segurança da macro estiver configurado para não permitir macros, os programas em um suplemento instalado ainda podem executar.
- Geralmente, as funções personalizadas funcionam somente na pasta de trabalho em que elas são definidas. Uma função personalizada adicionada a um suplemento está disponível para todas as pastas de trabalho abertas.
- O suplemento não aparece na lista de arquivos abertos no item de menu Janela. O cliente não pode reexibir a pasta de trabalho escolhendo Janela, Reexibir.

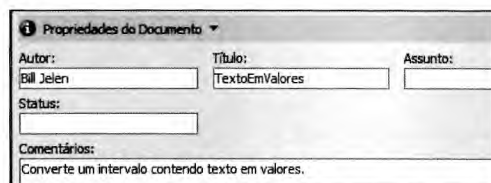
Há uma estranha regra para a qual você precisa se planejar. O suplemento é uma pasta de trabalho oculta. Como o suplemento nunca pode ser exibido, seu código não pode selecionar nem ativar nenhuma célula na pasta de trabalho de suplementos. É permitido salvar dados no arquivo de suplementos, mas você não pode selecionar o arquivo. Além disso, se você gravar os dados no arquivo de suplementos que você quer disponibilizar futuramente, os códigos de suplementos precisam tratar o salvamento do arquivo. Como seus clientes não saberão que o suplemento estará aí, eles nunca serão lembrados de, ou solicitados a salvar um suplemento não salvo. Você pode adicionar `ThisWorkbook.Save` ao evento `Workbook_BeforeClose` do suplemento.

Convertendo uma pasta de trabalho Excel em um suplemento

Os suplementos são em geral gerenciados pela caixa de diálogo Suplementos que apresenta um nome e uma descrição do suplemento. Você pode controlar esses inserindo duas propriedades específicas para o arquivo antes de convertê-lo em um suplemento.

Para alterar o título e a descrição mostrados na caixa de diálogo Suplementos, siga estes passos:

1. No ícone do Office, escolha Preparar, Propriedades. O Excel exibirá o painel Propriedades do Documento na parte superior da planilha.
2. Insira o nome para o suplemento no campo Título.
3. Insira uma breve descrição do suplemento no campo Comentários (veja Figura 27.1).
4. Clique no X no canto superior direito do painel Propriedades do Documento para fechá-lo.

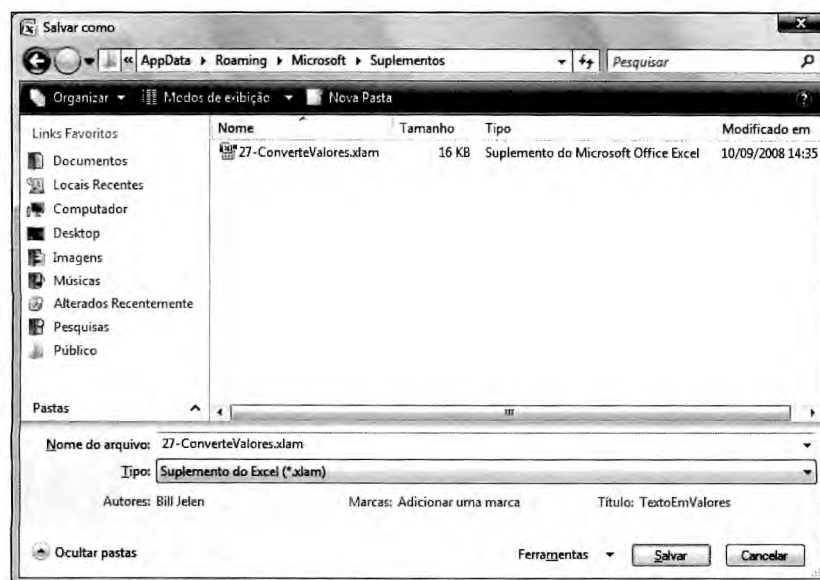
Figura 27.1 Preencha os campos Título e Comentários antes de converter uma pasta de trabalho em um suplemento.

Há duas maneiras de converter o arquivo em um suplemento. O primeiro método, utilizando Salvar Como, é mais fácil, mas tem um subproduto irritante. O segundo método utiliza o Editor do VB e requer dois passos, mas oferece algum controle extra. As seções que seguem descrevem os passos para utilizar esses métodos.

Utilizando Salvar Como para converter um arquivo em um suplemento

A partir do ícone do Office, selecione Salvar Como, Outros Formatos. No campo Salvar Como, Tipo, role pela lista e selecione Suplemento do Excel (*.xlam).

> **NOTA** Se o suplemento puder ser utilizado no Excel 97 através do Excel 2007, escolha Suplemento Excel 97-2003 (*.xla).

Como mostrado na Figura 27.2, o nome de arquivo muda de Algo.xlsm para Algo.xlam. Note também que a localização de salvamento muda automaticamente para uma pasta Suplementos. Essa localização de pasta varia pelo sistema operacional, mas será algo do tipo C:\Documents and Settings\Customer\Application Data\Microsoft\AddIns. Também é confuso que, depois de salvar o arquivo XLSM como um tipo XLAM, o arquivo XLSM não salvo permanece aberto. Não é necessário manter uma versão XLSM do arquivo, porque é fácil mudar novamente um XLAM para um XLSM para edição.

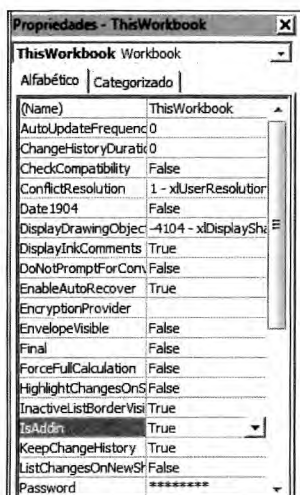
Figura 27.2 Se você estiver criando um suplemento para uso próprio, o método Salvar Como altera a propriedade `IsAddIn`, altera o nome e automaticamente salva o arquivo em sua pasta Suplementos.

ATENÇÃO

Ao utilizar o método Salvar Como para criar um suplemento, a planilha deve ser uma planilha ativa. O tipo de arquivo Suplementos não estará disponível se a planilha de Gráfico for a planilha ativa.

Utilizando o Editor do VB para converter um arquivo em um suplemento

O método Salvar Como é ótimo se você estiver criando um suplemento para uso próprio. Mas se o estiver fazendo para um cliente, você provavelmente irá querer manter o suplemento armazenado em uma pasta com todos os arquivos de aplicativo do cliente. É relativamente fácil pular o método Salvar Como e criar um suplemento que utiliza o Editor do VB:

1. Abra a pasta de trabalho que você quer converter em um suplemento.
2. Mude para o Editor do VB.
3. No Project Explorer, dê um clique duplo em ThisWorkbook.
4. Na janela Propriedades, localize a propriedade chamada `IsAddIn` e mude seu valor para `True`, como mostrado na Figura 27.3.

Figura 27.3
Criar um suplemento é tão simples quanto alterar a propriedade `IsAddIn` de ThisWorkbook.

5. Pressione Ctrl+G para exibir a janela Verificação Imediata. Nessa janela, salve o arquivo, utilizando uma extensão .xlam:

```
ThisWorkbook.SaveAs FileName:="C:\ClientFiles\Chap27.xlam", FileFormat:=xlAddIn8
```

NOTA

Se o suplemento for utilizado no Excel 97-2003, mude o parâmetro final xlAddIn8 para xlAddIn.

Agora você criou com sucesso um suplemento na pasta do cliente que pode ser facilmente localizada e enviada por e-mail para ele.

Fazendo o cliente instalar o suplemento

Depois de enviar o suplemento por e-mail para o cliente, faça com que ele o salve na área de trabalho ou em outra pasta fácil de localizar. O cliente deve então seguir estes passos:

1. Abrir o Excel 2007. A partir do menu do ícone do Office, selecionar Opções do Excel.
2. Ao longo da navegação esquerda, selecionar Suplementos.
3. Na parte inferior da janela, escolher Suplementos do Excel a partir da lista suspensa Gerenciar (veja Figura 27.4).
4. Clicar em Ir. O Excel exibirá a conhecida caixa de diálogo Suplementos.
5. Na caixa de diálogo Suplementos, clicar no botão Procurar (veja Figura 27.5).
6. Na caixa de diálogo Procurar, selecionar Desktop. Destacar o suplemento e escolher OK.

O suplemento é então instalado. O Excel copia o arquivo da área de trabalho para a localização adequada da pasta Suplementos. No caixa de diálogo Suplementos, o título do suplemento e os comentários conforme especificados na caixa de diálogo Propriedades do Arquivo são exibidos (veja Figura 27.6).

Figura 27.4
A guia Suplementos do Excel 2007 em Opções do Excel é significativamente mais complexa do que a do Excel 2003. Escolha Suplementos do Excel na parte inferior e clique em Ir.

Figura 27.5
Seu cliente seleciona Procurar a partir da caixa de diálogo Suplementos.

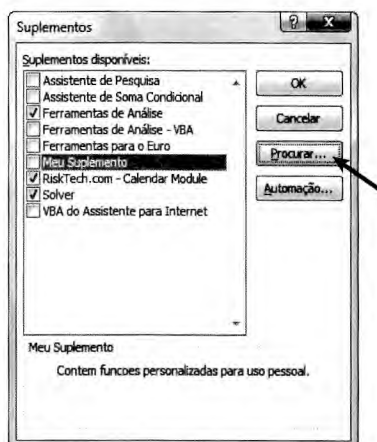
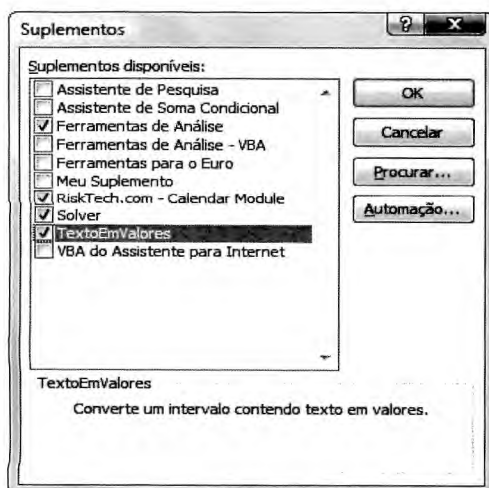



Figura 27.6
O suplemento agora está disponível para utilização.

Suplementos-padrão não são seguros

Lembre-se de que qualquer pessoa pode ir para o Editor do VB, selecionar o suplemento e mudar a propriedade `IsAddin` para `False` a fim de reexibir a pasta de trabalho. Você pode evitar que isso ocorra bloqueando o projeto XLAM para visualizá-lo e protegê-lo no Editor do VB, mas esteja ciente de que um grande número de fornecedores vende um utilitário de hacking de senha por menos de US$ 40. Para adicionar uma senha ao suplemento, siga estes passos:

1. Vá para o Editor do VB.
2. Do menu Ferramentas, selecione Propriedades de VBAProject.
3. Selecione a guia Proteção.
4. Marque a caixa Bloquear Projeto para Exibição.
5. Insira a senha duas vezes para confirmação.

Fechando os suplementos

Os suplementos podem ser fechados de três maneiras:

1. Desmarque o suplemento da caixa de diálogo Suplementos. Isso fecha o suplemento para essa sessão e assegura que ele não será aberto durante sessões futuras.
2. Utilize o Editor do VB para fechar o suplemento. No painel Verificação Imediata do Editor do VB, digite esse código para fechar o suplemento:

   ```
   Workbooks("YourAddinName.xlam").Close
   ```
3. Feche o Excel. Todos os suplementos serão fechados quando o Excel for fechado.

Removendo os suplementos

Talvez você queira remover um suplemento da lista de suplementos disponíveis na caixa de diálogo Suplementos. Não há nenhuma maneira efetiva de fazer isso dentro do Excel. Siga estes passos:

1. Feche todas as instâncias em execução do Excel.
2. Utilize o Windows Explorer para localizar o arquivo. Talvez o arquivo esteja em %AppData%\ Microsoft\AddIns\.
3. No Windows Explorer, renomeie o arquivo ou transfira-o para uma pasta diferente.
4. Abra o Excel. Você recebe uma nota advertindo-o de que o suplemento não pode ser localizado. Clique em OK para fechar essa caixa.
5. Utilize o ícone do Office, Suplementos, Gerenciar Suplementos do Excel, Ir. Na caixa de diálogo Suplementos, desmarque o nome do suplemento que você quer remover. O Excel notifica-o de que o arquivo não pôde ser localizado e pergunta se você gostaria de removê-lo da lista. Escolha Sim.

Utilizando uma pasta de trabalho oculta como alternativa a um suplemento

Um recurso interessante dos suplementos é que a pasta de trabalho é oculta. Isso impede que a maioria dos usuários iniciantes mexa nas fórmulas e modifique-as. Mas é possível ocultar uma pasta de trabalho sem criar um suplemento.

É muito fácil ocultar uma pasta de trabalho selecionando Ocultar do menu Window no Excel. O truque é então salvar a pasta de trabalho como Oculta. Como o arquivo está oculto, escolher o ícone comum do Office, Salvar, não funciona. Isso pode ser feito a partir da janela do Editor do VB. No Editor do VB, confirme se a pasta de trabalho está selecionada no Project Explorer. Então, na janela Verificação Imediata, digite o seguinte:

```
ThisWorkbook.Save
```

Estudo de caso

Utilizando uma pasta de trabalho de código oculta para armazenar todas as macros e formulários

Os desenvolvedores do Access usam com freqüência um segundo banco de dados para armazenar macros e formulários. Eles colocam todos os formulários e programas em um banco de dados e todos os dados em um banco de dados separado. Esses arquivos de banco de dados são vinculados pela função Link Tables no Access.

Para grandes projetos no Excel, recomendo o mesmo método. Você utiliza um pouco de código VBA na pasta de trabalho Dados para abrir a pasta de trabalho Código.

A vantagem desse método é que, quando for a hora de aprimorar o aplicativo, você pode enviar um novo arquivo de código por e-mail sem afetar o arquivo de dados do cliente.

Uma vez encontrei um aplicativo de um único arquivo lançado por outro desenvolvedor que o cliente tinha enviado para 50 representantes de vendas. Os representantes replicaram o aplicativo para cada um de seus 10 maiores clientes. Dentro de uma semana, havia 500 cópias desse arquivo espalhadas por todo o país. Quando eles descobriram um defeito importante no programa, corrigir esses 500 arquivos foi um pesadelo.

Projetamos um aplicativo de substituição que utilizava duas pastas de trabalho. A pasta de trabalho de dados acabou com aproximadamente 20 linhas de código. Esse código era responsável por abrir a pasta de trabalho de código e passar o controle para a pasta de trabalho do código. Enquanto os arquivos eram fechados, a pasta de trabalho de dados fecharia a pasta de trabalho de código.

Há muitas vantagens nesse método. Primeiro, os arquivos de dados do cliente foram mantidos em um tamanho muito pequeno. Todos os representantes de vendas agora têm uma pasta de trabalho com o código do programa e dez ou mais arquivos de dados para cada cliente. À medida que os aprimoramentos são completados, distribuímos novas pastas de trabalho de código do programa. Os representantes de vendas abrem sua pasta de trabalho de dados de cliente existente, que automaticamente vem com a nova pasta de trabalho de código.

Como o desenvolvedor anterior ficou empacado com o trabalho de tentar corrigir 500 pastas de trabalho, fomos extremamente cuidadosos para que houvesse o menor número possível de linhas de código na pasta de trabalho. Pode haver dez linhas de código e elas foram extremamente testadas antes de serem enviadas. Em contraste, a pasta de trabalho de código contém mais de 3 mil linhas de código. Então, se algo sair errado, tenho uma chance de 99 por cento de que o código ruim estará na pasta de trabalho de código fácil de substituir.

Na pasta de trabalho de dados do cliente, o procedimento `Workbook_Open` tem esse código:

```
Private Sub Workbook_Open()
    On Error Resume Next
    X = Workbooks("Code.xls").Name
    If Not Err = 0 then
        On Error Goto 0
        Workbooks.Open Filename:=ThisWorkbook.Path & Application.PathSeparator & "Code.xlsm"
    End If
    On Error Goto 0
    Application.Run "Code.xls!CustFileOpen"
End Sub
```

O procedimento `CustFileOpen` na pasta de trabalho de código trata de adicionar um menu personalizado ao aplicativo. Esse procedimento também chama uma macro chamada `DeliverUpdates`. Se você precisasse alguma vez alterar os 500 arquivos de dados de cliente, a macro `DeliverUpdates` trataria disso via código.

Esta solução de pasta de trabalho dual funciona bem e permite que as atualizações sejam entregues com transparência para o cliente sem tocar em nenhum dos 500 arquivos dele.

Próximos passos

Se como autores fizemos nosso trabalho corretamente, você agora tem as ferramentas necessárias para projetar seus próprios aplicativos VBA no Excel. Você entende as falhas do programa de gravação de macros, sabendo utilizá-lo como uma ajuda ao aprender a fazer algo. Sabe utilizar as poderosas ferramentas do Excel no VBA para produzir rotinas eficientes que podem economizar muitas horas de trabalho por semana. Você também aprendeu a fazer um aplicativo interagir com outros a fim de criar aplicativos para serem utilizados pela sua ou por outras organizações.

Se você achou alguma seção do livro confusa ou que poderia ter sido mais bem escrita, seremos gratos por seus comentários e eles serão considerados na preparação da próxima edição deste livro. Escreva-nos: Bill@MrExcel.com e Tracy@MrExcel.com.

Se o seu objetivo era automatizar algumas tarefas próprias ou tornar-se um consultor em Excel, esperamos tê-lo ajudado. Ambos são objetivos recompensadores. Com 500 milhões de clientes potenciais, achamos que prestar consultoria em Excel é um bom negócio. Se estiver interessado em juntar-se às nossas fileiras, este livro é seu manual de treinamento. Domine os tópicos e estará qualificado para juntar-se à equipe de consultores do Excel.

Índice

Números e símbolos

A

D

E

F

G

M

N

O

P

Q

R

S

W

X

Z